UNIVERSIDAD AUTÓNOMA DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

Departamento de Tecnología Electrónica y de las Comunicaciones

Ph.D. Thesis

On the Exploration of FPGAs and High-Level Synthesis Capabilities on Multi-Gigabit-per-Second Networks

MARIO DANIEL RUIZ NOGUERA

Ph.D. Advisors:
Dr. Gustavo Sutter
Dr. Sergio López Buedo

MADRID, OCTOBER 2019

Francisco Tomás y Valiente, 1
Madrid, 28049
Spain

MARIO DANIEL RUIZ NOGUERA
On the Exploration of FPGAs and High-Level Synthesis Capabilities on Multi-Gigabit-per-Second Networks

Escuela Politécnica Superior.
High Performance Computing and Networking Research Group

IMPRESO EN ESPAÑA — PRINTED IN SPAIN

Department: Tecnología Electrónica y de las Comunicaciones
Escuela Politécnica Superior
Universidad Autónoma de Madrid

Ph.D. Thesis: On the Exploration of FPGAs and High-Level Synthesis Capabilities on Multi-Gigabit-per-Second Networks

Author: Mario Daniel Ruiz Noguera

Advisors: Dr. Gustavo Sutter

Dr. Sergio López Buedo

Year: 2019

Committee:

President	Prof. Dr. Gustavo Alonso
Secretary	Dr. Iván González Martínez
Member	Dr. Giulio Gambardella
Member	Dr. Gianni Antichi
Member	Dr. Zsolt István
Substitute Member	Prof. Dr. Javier Aracil Rico
Substitute Member	Dr. Raúl Mateos Gil

This thesis was carried out within the High Performance Computing and Networking research group, at the Departamento de Tecnología Electrónica y de las Comunicaciones, Escuela Politécnica Superior, Universidad Autónoma de Madrid. Part of this thesis was carried out while doing an academic visit at the Systems Group of the ETH Zürich, Switzerland. This thesis was partially supported by the Spanish Government (grants MINECO/FEDER: TEC2012-33754 and TEC2015-69417-C2-1-R); and by the European Union through the Integrated Project FP7-317999 and grant agreements 687632 and 761727 of the H2020 program. And by the Departamento de Tecnología Electrónica y de las Comunicaciones, Escuela Politécnica Superior, Universidad Autónoma de Madrid. Furthermore, Mario Daniel Ruiz Noguera was funded in part by the Universidad Autónoma de Madrid through an FPI-UAM (Research Staff Training) scholarship from November 2016 until the defense of this thesis.

A mis padres y hermana: Piedad, Mario y Florencia, por recorrer esta travesía conmigo.
A Silvia e Alberto, per la vostra amicizia.

Abstract

Traffic on computer networks has faced an exponential grown in recent years. Both links and communication equipment had to adapt in order to provide a minimum quality of service required for current needs. However, in recent years, a few factors have prevented commercial off-the-shelf hardware from being able to keep pace with this growth rate, consequently, some software tools are struggling to fulfill their tasks, especially at speeds higher than 10 Gbit/s. For this reason, Field Programmable Gate Arrays (FPGAs) have arisen as an alternative to address the most demanding tasks without the need to design an application specific integrated circuit, this is in part to their flexibility and programmability in the field. Needless to say, developing for FPGAs is well-known to be complex. Therefore, in this thesis we tackle the use of FPGAs and High-Level Synthesis (HLS) languages in the context of computer networks. We focus on the use of FPGA both in computer network monitoring application and reliable data transmission at very high-speed. On the other hand, we intend to shed light on the use of high level synthesis languages and boost FPGA applicability in the context of computer networks so as to reduce development time and design complexity.

In the first part of the thesis, devoted to computer network monitoring. We take advantage of the FPGA determinism in order to implement active monitoring probes, which consist on sending a train of packets which is later used to obtain network parameters. In this case, the determinism is key to reduce the uncertainty of the measurements. The results of our experiments show that the FPGA implementations are much more accurate and more precise than the software counterpart. At the same time, the FPGA implementation is scalable in terms of network speed — 1, 10 and 100 Gbit/s. In the context of passive monitoring, we leverage the FPGA architecture to implement algorithms able to thin cyphered traffic as well as removing duplicate packets. These two algorithms straightforward in principle, but very useful to help traditional network analysis tools to cope with their task at higher network speeds. On one hand, processing cyphered traffic bring little benefits, on the other hand, processing duplicate traffic impacts negatively in the performance of the software tools.

In the second part of the thesis, devoted to the TCP/IP stack. We explore the current limitations of reliable data transmission using standard software at very high-speed. Nowadays, the network is becoming an important bottleneck to fulfill current needs, in particular in data centers. What is more, in recent years the deployment of 100 Gbit/s network links has started. Consequently, there has been an increase scrutiny of how networking functionality is deployed, furthermore, a wide range of approaches are currently being explored to increase the efficiency of networks and tailor its functionality to the actual needs of the application at hand. FPGAs arise as the perfect alternative to deal with this problem. For this reason, in this thesis we develop Limago an FPGA-based open-source implementation of a TCP/IP stack operating at 100 Gbit/s for Xilinx's FPGAs. Limago not only provides an unprecedented throughput, but also, provides a tiny latency when compared to the software implementations, at least fifteen times. Limago is a key contribution in some of the hottest topic at the moment, for instance, network-attached FPGA and in-network data processing.

Resumen

El tráfico en las redes de ordenadores ha crecido exponencialmente durante los últimos años. Tanto los enlaces como los equipos de comunicación han tenido que adaptarse para proveer la calidad de servicio mínima dependiendo de la aplicación. Sin embargo, en los últimos años, ciertos factores han impedido que el *hardware* tradicional sea capaz de seguir este ritmo de crecimiento, por lo tanto, algunas herramientas tienen problemas para cumplir sus tareas, especialmente a velocidades mayores a 10 Gbit/s. Por este motivo, las FPGA surgen como una alternativa para implementar las tareas más demandantes sin tener que desarrollar un circuito específico, esto es debido en parte a su gran flexibilidad y alta capacidad de programación en el campo. Sin embargo, el desarrollo en esta tecnología es conocido por ser complejo. Es por ello que en esta tesis abordamos el uso de FPGAs y lenguajes de alto nivel en el ámbito de redes de ordenadores. Tanto en la monitorización de las mismas, como en la transmisión fiable de datos a muy alta velocidad. Mientras que con el uso de lenguajes de alto nivel pretendemos posicionarlos como una alternativa para reducir el tiempo y complejidad de desarrollo en el contexto de redes de ordenadores.

En la primera parte de esta tesis, dedicada a la monitorización de redes de ordenadores. Aprovechamos del determinismo de las FPGAs para implementar sondas de monitorización activas, las cuales consisten en enviar un tren de paquetes a partir del cual se pueden medir parámetros de red. En este caso el determinismo es primordial para reducir la incertidumbre en las medidas. Los resultados de los experimentos muestran que la implementación FPGA es mucho más exacta además de tener mayor precisión en las medidas que la versión *software*. A su vez, la implementación FPGA es escalable a diferentes velocidades de enlace — 1, 10 y 100 Gbit/s. En el ámbito de la monitorización pasiva, también aprovechamos de las FPGAs para implementar algoritmos capaces de reducir el tráfico encriptado y eliminar paquetes duplicados. Estos dos algoritmos, simples en principio, pero útiles a la hora de ayudar a las herramientas tradicionales de análisis de red a seguir operando en enlaces de mayor velocidad. Por un lado, analizar tráfico encriptado no trae mayores beneficios, mientras que analizar tráfico duplicado impacta negativamente en el desempeño de las herramientas de *software*.

En la segunda parte de esta tesis, dedicada al protocolo TCP/IP. En el contexto actual, las redes se están convirtiendo en un cuello de botella para satisfacer las necesidades modernas, en particular en los centros de datos. A demás en los últimos años se comenzó a desplegar enlaces de 100 Gbit/s. En este sentido, hay un mayor escrutinio de cómo se implementa la funcionalidad de red, además, se están explorando una gran variedad de alternativas para incrementar la eficiencia y adaptar su funcionalidad a las necesidades reales de la aplicación en cuestión. Las FPGAs se presentan como la alternativa ideal para solventar este problema. Es por esto que en esta tesis desarrollamos Limago una implementación de código abierto la cual implementa el protocolo TCP/IP completo a 100 Gbit/s para las FPGAs de Xilinx. A demás de lograr un ancho de banda sin precedentes en transmisión fiable de datos, también, presenta una latencia muy baja comparada con las implementaciones de *software*, al menos quince veces. Limago es una contribución de vital importancia en algunos de los temas de investigación más activos en el momento, por ejemplo, FPGAs conectadas directamente a la red y procesamientos de datos en la fuente de los mismos.

Agradecimientos

Esta tesis es el esfuerzo de los últimos cuatro años de mi vida. Haber transitado este camino no habría sido posible sin la ayuda y soporte de muchas personas, algunas de las cuales están desde el principo otras estuvierón durante parte del camino mientras que otras ya no están entre nosotros. Por este motivo quiero dedicar unas líneas para mostrar mi gratidud.

Primero que nada, quiero agradecer a mi familia, mi madre Piedad, mi padre Mario y mi hermana Florencia que a la distancia proporcionaron invaluable soporte durante estos años. Seguramente sin su soporte, el camino habría sido mucho más duro de lo que es. De la misma forma quiero agradecer al resto de integrantes mi familia por su soporte y comprensión.

Quiero reconocer el trabajo y esfuerzo de mis directores de tesis, Gustavo Sutter y Sergio López Buedo. Su subiduría y guía ha sido fundamental para lograr los objetivos de esta tesis y así llegar hasta este punto. No sólo han sido mi guía en los caminos de la investigación, sino que también han sido buenos amigos. En la misma línea quiero agradecer al resto de profesores del laboratorio HPCN, Iván, Paco, Luis de Pedro, Javier Aracil, Javier Ramos, José Luis y Lluís, pero especialmente a Jorge López de Vergara el cual fue una gran fuente de inspiración.

También quiero agradecer a todos mis compañeros de laboratorio en estos casi cinco años: Rafael, José Fernando, David, Paula, Carlos, Ángel, Jesús, Anotonis, Dixon, Sergio, Dani, Eduardo, Rubén y Luis. Hemos compartido muchas horas en el laboratorio y muchos momentos han sido memorables. Quiero agradecer especialmente a Tobías, con el cual siempre hemos tenido discusiones productivas las cuales me ayudaron tanto a solucionar problemas como así también a crecer como investigador.

Finalmente, quiero agradecer a mi familia en Madrid Alberto y Silvia. La vida fuera del laboratorio no habría sido igual sin ustedes. Su compañía en muchas ocasiones ha puesto una sonrisa en mi cara.

Additionally, I would like to thank Gustavo Alonso for allowing me to collaborate with his group (Systems Group at the ETHz). Limago emerged from such collaboration and it has been proven to be one of the major contribution of this thesis. During my stay at ETHz I was able to meet great people David, Zsolt and Zeke.

Also, I would like to thank Michaela Blott, Giulio Gambardella, Lucian Petrica and the rest of the team of the Xilinx Research Labs for the great opportunity of doing an internship with them. Moreover, during my internship at Xilinx I meet extraordinary people, Cathal, Gary, Francesco, Cristoph and Jorn.

List of Contents

Part II: Network Monitoring

Part III: Offload Tasks

List of Tables

List of Figures

List of Acronyms

AMBA Advanced Microcontroller Bus Architecture. 30

API Application Programming Interface. 8

ARP Address Resolution Protocol. 125–127, 131

ASCII American Standard Code for Information Interchange. 73–78, 146, 154

ASIC Application Specific Integrated Circuit. 5, 13, 24, 29, 116, 117

AXI4 Advanced eXtensible Interface 4. 30

BERT Bit Error Rate Test. 62

CAM Content-Addressable Memory. 84

COTS Commercial Off-The-Shelf. 4, 7, 49, 51, 69

CSA Carry Save Adder. 79, 107, 108, 110, 114, 121, 143, 151

DDS Direct Digital Synthesizer. 54

DUT Device Under Test. 10

EDA Electronic Design Automation. 26

EDC Electronic Dispersion Compensation. 57

FEC Forward Error Correction. 62

FIFO First-In First-Out. 76, 77, 88, 127, 145, 153

FLOPS Floating Point Operations per Second. 6, 14

FPGA Field Programmable Gate Array. i–iv, 3–11, 13–19, 21, 24–26, 29, 31, 36, 43–49, 52, 54, 55, 61, 62, 64, 66, 68, 69, 71, 73, 74, 81, 83, 85, 86, 91, 93, 98, 99, 106, 107, 116, 117, 124, 125, 132, 136–138, 141–144, 146, 149–152, 155

FSM Finite State Machine. 27, 62, 78, 128, 131

GP-GPU General-Purpose Graphics Processing Unit. 6, 14, 68

GPS Global Positioning System. 46, 47, 52, 54, 55, 68

HBM High Bandwidth Memory. 7, 15, 74, 138, 146, 155

HDL Hardware Description Language. 5, 14, 45, 52–55, 61, 68, 72, 78, 87, 89, 93, 114, 121, 137, 142–145

HLS High-Level Synthesis. i, 5–10, 14, 17–19, 27, 28, 44, 45, 47, 52–55, 61, 64, 68, 69, 72–77, 83, 86–89, 93, 98, 114, 118, 122, 123, 125, 127, 137, 141–146

ICMP Internet Control Message Protocol. 98, 99, 125, 127, 131

IP Internet Protocol. ii, iv, 3, 8–10, 17–19, 97–104, 114, 116–120, 124, 125, 127, 131, 132, 134, 138, 143, 144, 147, 151–153, 155

ISP Internet Service Provider. 22, 48

KPI Key Performance Indicators. 26, 44, 49

LTL Lightweight Transport Layer. 117

MAC Media Access Control. 29, 33, 46, 125–127

MGT Multi Gigabit Transceiver. 29, 57

MSS Maximum Segment Size. 124

MTU Maximum Transmission Unit. 56, 100, 103

NAT Network Address Translation. 100

NFV Network Function Virtualization. 116

NIC Network Interface Card. 8, 44, 51, 55, 68, 98–100, 116, 117, 124, 135, 138

OSI Open Systems Interconnection. 29, 97, 102, 118, 125

OWD One-Way Delay. 26, 44, 47, 50–52, 55, 56

PAL Programmable Array Logic. 24

PLR Packet Lost Ratio. 44, 50, 51

PPS Pulse Per Second. 54, 55

PTP Precision Time Protocol. 52

QoR Quality of Result. 5, 6, 122, 144–146

QoS Quality of Service. 10, 22, 44, 48, 49, 61, 141

RDMA Remote Direct Memory Access. 116

RSS Receive Side Scaling. 116

RTL Register Transfer Level. 5, 6, 14, 24, 26–28, 52, 99, 105, 114, 124, 144, 146, 153

RTT Round-Trip delay Time. 26, 44, 45, 50, 61, 64, 65, 122, 126

SDN Software Defined Network. 5, 13, 49

SLR Super Logic Region. 93

SoC System on a Chip. 30, 45, 47, 52, 68

SPAN Switched Port Analyzer. 82

SSL Secure Sockets Layer. 74

TCAM Ternary Content-Addressable Memory. 84

TCP Transmission Control Protocol. ii, iv, 8–10, 17–19, 97–103, 105, 113, 114, 116–120, 122, 124, 125, 127–132, 135, 138, 143, 144, 146, 147, 151–153, 155

TOE TCP Offload Engine. 8, 116, 122, 124–127, 131, 132, 136, 137, 145, 153, 154

UDP User Datagram Protocol. 97–99, 101–103, 105, 113, 117, 125

VHDL Very High Speed Integrated Circuit Hardware Description Language. 24, 25, 54, 73, 78, 79, 105

XAUI 10G Attachment Unit Interface. 29, 57

Part I

Introduction

CHAPTER 1

INTRODUCTION

This chapter furnishes the context of this Ph.D. Thesis. First of all, we delineate the pivotal motivation of this thesis, challenges and insights that guided its progress. Shortly after, we set in context this work in regards with the current trends in the FPGA and monitoring arena as well as FPGAs as network-attached elements. With that in mind, we formulate the objectives of this Thesis. Finally, we outline the Thesis organization and detail the content of each chapter.

1.1 Motivation of the Thesis

Our day-to-day activities are becoming more and more dependent on communication networks: the Internet, mobile apps, e-commerce, cloud applications, among others. Such widespread use of communications has implications at both access and server sides of the computer networks, for instance, there is an increasing need for fast and reliable networks. At the access side, it drives the development of faster access technologies, such as 10 Gbit/s passive optical networks or five generation (5G) mobile networks. At the server side, it causes exponential increase in network traffic being faced by data centers. From this perspective, the authors in the paper [1] argue that the exponential growth would not have been possible without exponential growth in the computing ecosystem: chip level, system level and adopting community. As they stated these factors feed themselves to keep the growth at an exponential pace. In this regard, Cisco forecasts [2] the IP traffic growth in the period of 2017 to 2022, see Figure 1.1. What is more, Lord

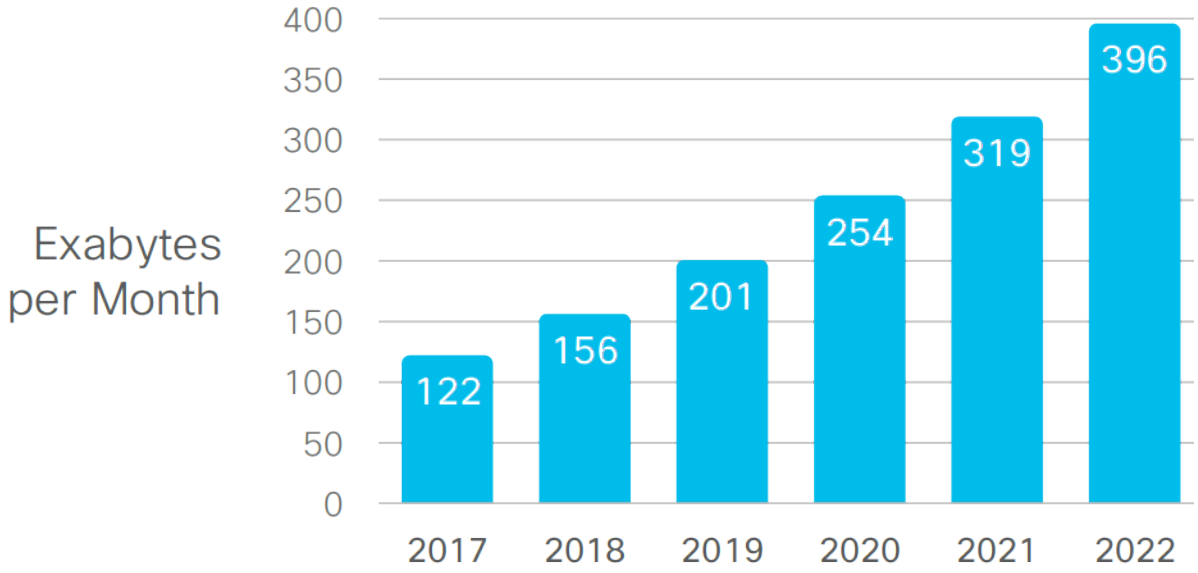


Figure 1.1: IP traffic compound annual growth rate. Source: [2].

et.al [3] explore the different factors that impact on the continuous growth on the Internet, for instance, Figure 1.2 shows the exponential growth in the broadband demand in the United Kingdom at peak time for a ten-year period. This figure confirms Nielsen's Law [4] of the Internet bandwidth, an empirical observation which states that *"A high-end user's connection speed grows by 50% per year"*.

It is expected that the network speed continues growing to meet the data traffic growth, thereupon, the computing performance should match the growing rate. In such a way, computer networks have turned into a critical infrastructure, where malfunctions cannot be tolerated. Hence, it is paramount to guarantee the quality of network links in order to ensure an appropriate operation of the whole ecosystem. As a result, network testing is key to assess such networks and it has become more necessary than ever before. Notwithstanding, network testing becomes a complex and expensive task at such speeds; with Dennard scaling dead (Figure 3.3), the traditional software-based approaches struggle to monitor multi-gigabit-per-second networks. Lately, application specific circuits to tackle network monitoring at such speed have gained traction so as to overcome the limitations of Commercial Off-The-Shelf (COTS) hardware.

Field Programmable Gate Array (FPGA) has proven to be able to perform a wide variety of monitoring task. Yet, in 2005 Taylor *et al.* [5] include a FPGA implementation in their survey for packet classification. In another survey in 2010 [6] the authors made a thorough and comprehensive research of the different approaches with FPGA for network security, they argue that for those applications surveyed the FPGA-based implementation surpassed the performance of the software-based counterpart. Moreover, the company Arista [7] in 2018 released a white paper with the four key trend in

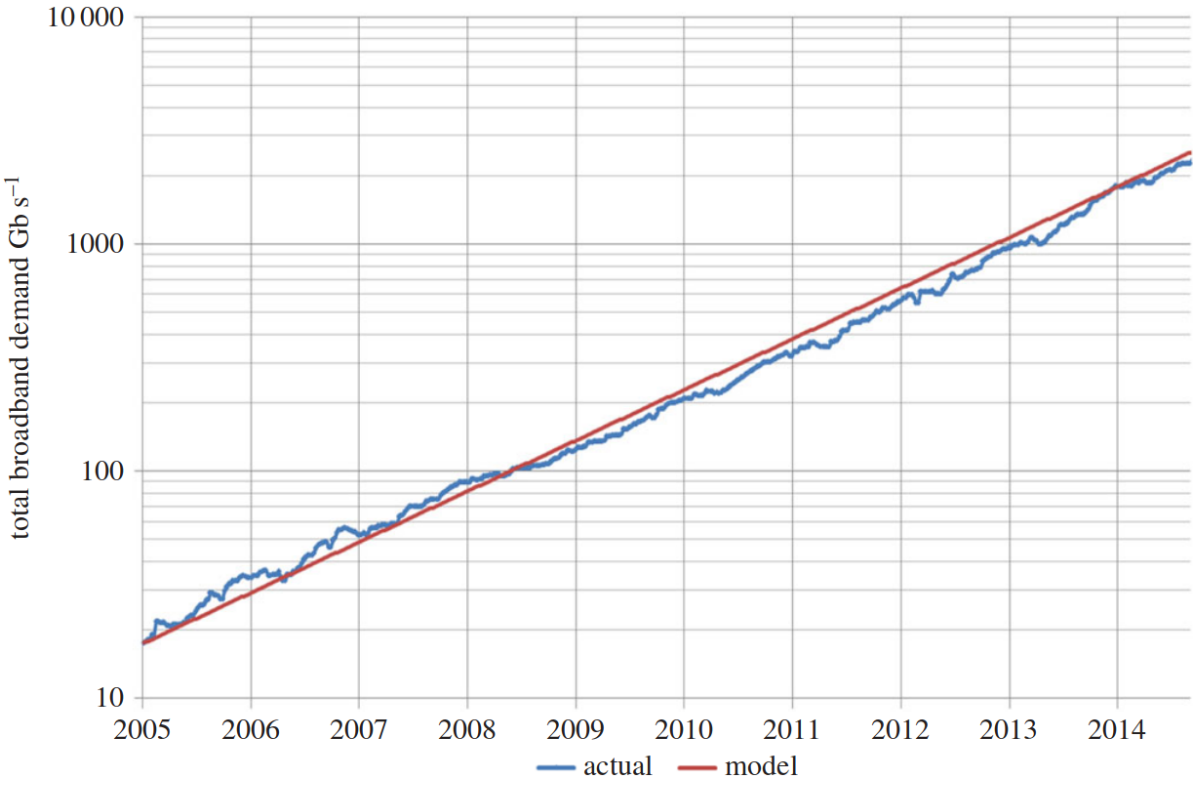


Figure 1.2: Total broadband demand for one ISP in the UK, measured at peak time over a ten-year period. Source: [3].

the networked use of FPGAs — Software Defined Network (SDN); latency-sensitive automated trading; network capture and timestamping; networked video. Therefore, FPGAs have arisen as a ubiquitous technology not only for network monitoring, but also, for other networking tasks, for instance, switching [8]. When compare to traditional Application Specific Integrated Circuit (ASIC), FPGA provides more flexibility and less non-recurrent engineering cost, while keeping the high degree of parallelism. In this light, FPGAs stand as one of the most advantageous and cost effective solutions to deal with challenging network monitoring. Furthermore, one of the driving forces of FPGA in networking has been the open-source hardware and software project NetFPGA [9–11]. It started in 2006 as a teaching project in Stanford University, and rapidly, it became a popular prototyping platforms for research and industry projects.

On the other hand, it is well-known that the FPGA development is lengthily and complex. For many years Hardware Description Languages (HDLs) have dominated how to describe programmable logic for FPGA designs. Nevertheless, in the last decade High-Level Synthesis (HLS) tools have demonstrated enough maturity as well as good Quality of Results (QoRs), consequently, they have gained a portion in FPGA development phase [12]. Briefly, HLS tools aims at enabling to describe hardware using languages such as C/C++ (a higher degree of abstraction when compared to the RTL paradigm). On

this subject, in the paper [13] the authors present an exhaustive survey of HLS tools, where they argued that, even though the QoR of HLS is not as good as handcrafted Register Transfer Level (RTL), the productivity can be over four times higher. Yet, in 2011 Cornu *et al.* [14] presented efficient hardware accelerator implementations of algorithms that gave better performance than the RTL counterpart. Forconesi *et al.* [15] present an evaluation of Vivado-HLS (Xilinx's commercial tool) implementing a flow monitoring application, they claim a reduction of one order of magnitude in the development time when compared to RTL design. Moreover, one of the key of the success of HLS tools, apart from the fact that the main programming language is C/C++, is the ability to explore the design space much quicker and usually without requiring to modify the source code. Furthermore, in the last lustrum the industry and research community introduced several tools for packet processing targeting FPGA designs [16–21] based on high level abstraction languages. These factors have fueled the use of HLS tools and FPGAs in the context of computer networks. Consequently, high level synthesis for networking applications is an active research topic.

Not only have FPGAs been used for networked tasks, but also, some researchers have used them as a compute node in a distributed environment [22–24]. When compare to General-Purpose Graphics Processing Unit (GP-GPU), FPGAs have less Floating Point Operations per Second (FLOPS); however, they are more flexible to map irregular algorithms. What is more, the communication latency is smaller due to the direct communication of the interface with the programmable logic [25]. In light of these features, in the last five years there have been efforts to shift the CPU-attached paradigm to a network-attached one [26–29]. This phenomenon has been driven by the growth of computation power and heterogeneity on FPGAs. The idea is to detach the FPGA from a host machine and communicate directly through the network. Thus, increasing the overall efficiency and reducing the communication overhead. This would not have been possible without an efficient and scalable 10 Gbit/s TCP/IP implementation which was made open-source back in 2015 [30]. The efforts in the community are focused on improving the abstraction model of the infrastructure. Furthermore, one of the current endeavors is to accelerate heavy tasks by moving the compute closer to the data so as to reduce the communication overhead [31]. However, there has been little effort on enhancing the underlying communication infrastructure to tailor it to today's needs. For instance, convolutional neural networks need a large bandwidth to communicate the convolutional layers.

1.2 Context of this Thesis

As stated above, the COTS (plus it associated software) solutions for monitoring high-speed networks are struggling to keep the pace of the network link evolution. Thus, FPGAs have emerged as a ubiquitous technology to tackle some of the most complex tasks. Needless to say, FPGA development has been considered a skillfulness labor. However, FPGA design is evolving towards HLS which, not only reduces the development time, but also, democratizes the use of FPGAs owing to a simpler design methodology. On the other hand, as every technology, FPGAs have their limitations, for instance, large memory requirements with low-latency access. Nonetheless, the new families include harden High Bandwidth Memory (HBM) that might help to overcome such limitation.

In this regard, network monitoring; FPGA for networking and HLS described above have drawn the attention of the community for the last years. Remarkably, much of the research is contemporary to the period of this thesis. Figure 1.3 plots the number of references that can be found on the Web of Science (WoS) for every year in the period of 1990 to 2018 for the following queries:

- FPGA and network* NOT (“neural network*”)
- NetFPGA
- FPGA and “High Level Synthesis”

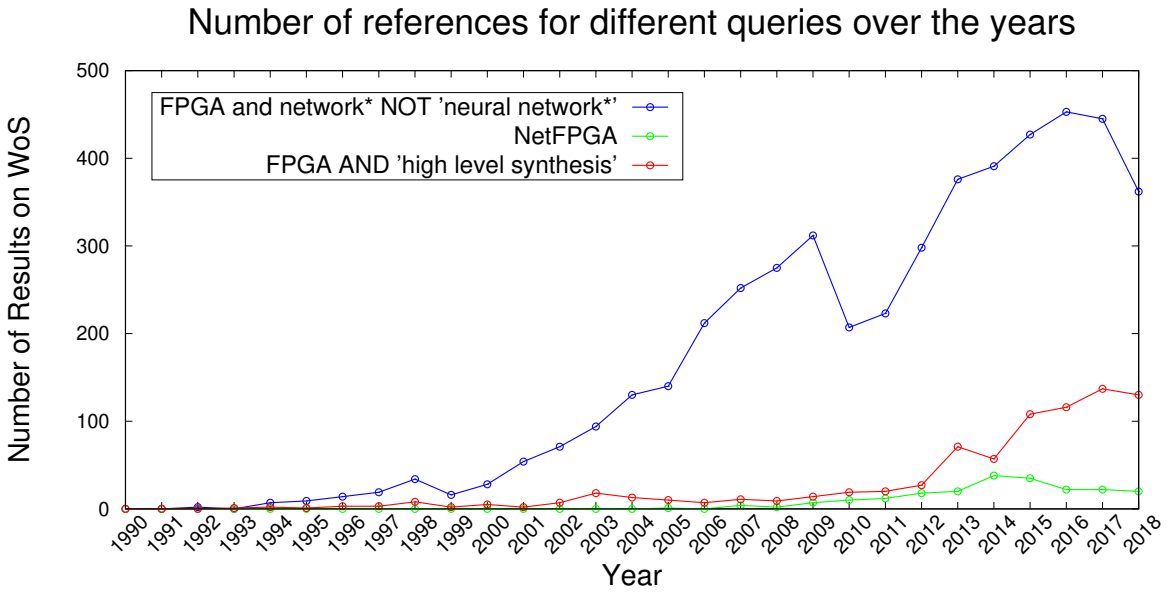


Figure 1.3: Progression of number of published works available from the Web of Science for each query, period 1990-2018.

The plot shows an almost linearly rise of results in monitoring with FPGAs over the years, since 1999. This fact confirms that FPGAs have gained a portion of the academic interest in that matter, by virtue of their myriad applicability. Despite that, most of the research has been focused on security and deep packet inspection. The NetFPGA brought light into packet processing; switching schemes and some testing appliances. Antichi *et al.* [32] were one of the pioneers in passive monitoring system using NetFPGA. In that work they presented a system based on the cooperation of a NetFPGA and a general purpose host machine. As a follow up, the paper [33] presents OSNT: an open-source network tester, with packet generation as well as packet capture capabilities. Likewise, Oeldemann *et al.* [34] present a network tester which is similar to the OSNT work but with subtle improvements on the trace replay and software programmability through their Application Programming Interface (API). What is more, Puš *et al.* in 2015 presented a custom board with an FPGA able to handle 100 GbE links, the FPGA distributes the traffic into different cores in the host machine for further processing.

With regard to HLS, the research has been mainly focused on accelerating algorithms, especially machine learning accelerators. In spite of that, there have been efforts towards monitoring appliances using HLS [15, 35]. The introduction of P4 [36], a domain-specific language optimized around network data, led the way of efficient and high level abstraction models for packet processing [17–19, 37–39]. In 2017, Emu [40] was introduced to enable rapid development of network functionality. What is more, Eran *et al.* [20] presented a library for packet processing aiming at code reusability, which is built on top of HLS.

Additionally, the increasing demands for distributed computing and the performance and scalability of many distributed applications depend upon an efficient implementation of the Transmission Control Protocol (TCP) and the Internet Protocol (IP). Implementing a TCP Offload Engine (TOE) on an FPGA opens interesting opportunities to explore programmable Network Interface Cards (NICs) as they offer higher flexibility than many commercial NIC. Unfortunately, reaching 100 Gbit/s speed is challenging, even using NIC-based TOE. Hence, Sidler *et al.* [30] paved the way in 2015 with a scalable 10 Gbit/s TCP/IP full stack implemented on an FPGA. Thus, most of the network-attached FPGA works have been built on top of it [26, 41].

All in all, this plot acknowledges the increase efforts of the community on FPGA for networking, as well as the raised of HLS efforts. On the other hand, the number of NetFPGA results have stalled since 2015, this is mainly because the latest version of the platform has not been updated since 2014.

1.3 Objectives of this Thesis

Previously, we have presented the research lines of this thesis. Consequently, in this section we describe the main objectives of this thesis. We lay out three main objectives. The first one delves deep into the FPGA capabilities for network monitoring in both active and passive areas. The second one seeks for a very high-speed low-latency and reliable data transmission implementation for distributed FPGA infrastructure, where FPGAs are detached from host machines and connected directly to the network. Finally, the third one scrutinizes the applicability of HLS in the FPGA design methodology for networked applications, this objective is fully embedded in the two previous objectives.

The following items summarizes the sub-objectives of this Thesis:

- **Explore FPGA capabilities in active monitoring**: FPGAs are well-known for their high degree of parallelism as well as their determinism. We want to take advantage of these two factors to perform active measurement. In particular, we want to contrast the accuracy of FPGA-based solutions against the software-based counterpart using the packet-train technique.

- **Explore FPGA capabilities in passive monitoring**: in this regard, we want to leverage the heterogeneity of FPGAs to thin traffic smartly without losing any relevant information. We explore how to cap cyphered packets, and how to remove duplicate packets. Aiming at helping traditional software tools not to handle unnecessary data.

- **Explore FPGA capabilities to implement a full TCP/IP stack**: the newer paradigm of host machine detached FPGA (network-attached) calls for an efficient communication protocol. TCP has been the *de facto* standard of reliable data transmission, in spite of that, it is well-known for being a very demanding due to its high memory consumption to keep track of the state of each connection. We want to push further the FPGA capabilities and provide the community with an open-source TCP/IP implementation at 100 Gbit/s with very low-latency, where they can build their own application on top of it.

- **Evaluate the applicability of HLS**: this thesis presents a unique opportunity to evaluate plain HLS (C/C++) in the context of a wide variety of networking applications. Therefore, we try applying HLS as much as possible in our development. In particular, we use Vivado-HLS the commercial solution of Xilinx.

This thesis is highly experimental. Therefore, each objective has been verified with a proof-of-concept implementation, and, in some cases we propose more than one solution.

1.4 Thesis Structure

To conclude the introduction, this section summarizes the rest of this document. This Thesis is divided into four parts, **Part I Introduction**: Chapter 3 presents a technical background on the different technologies used in the development of this thesis. It also provides a brief history of the integrated circuits, computer networks and FPGAs. Further, we present some concepts about network monitoring. The main idea of this chapter is to put the reader in context of the different technologies that clash in this Thesis. Finally, it shows every of each FPGA platforms used during this Thesis.

Part II Network Monitoring: in this part the we explore the FPGA capabilities to perform network monitoring. In particular Chapter 4 focuses on active monitoring, where we take advantage of the FPGA determinism to implement the packet-train technique in order to assess the Quality of Service (QoS) of a Device Under Test (DUT). The scalability of the above mentioned technique, as well as its FPGA implementation, has been proven with the assessment of different network link speed — 1, 10 and 100 Gbit/s. Secondly, Chapter 5 aims for providing bump-in-the-wire implementations to thin the traffic in order to help traditional monitoring tools to cope with their task. We explore alternatives to do so without losing relevant information. We present two designs where ciphered packets are capped and duplicate packets are removed. In doing so, we explore the heterogeneity of the state-of-the-art FPGAs.

In **Part III Offload Tasks**, we explore the FPGA architecture to implement very demanding tasks, but this time aiming for offloading part or the whole TCP/IP stack to a single FPGA. We first explore different circuits to offload the 16-bit one's complement checksum widely used in the IPv4 in Chapter 6. In Chapter 7 we present Limago an FPGA-based 100 GbE TCP/IP stack, which is based on a previous work. Limago tenfolds the communication speed when compared with the starting point while keeping its scalability in term of concurrent connections. To do so, we have widely used HLS not only to accelerate the development process, but also, to gain flexibility. This work contributes with a step forward in the widespread of FPGAs into distributed environments.

Finally, in **Part IV Conclusions**, Chapters 8 and 9 in English and Spanish respectively summarize the conclusion and finals remarks of this thesis. In addition to that, we present the contributions to the community, a discussion whether it makes sense to use HLS and lastly future directions in these topics.

CHAPTER 2

INTRODUCCIÓN

Este capítulo provee el contexto de esta tesis doctoral. Primero, presentamos las motivaciones esenciales de esta tesis, desafíos y percepciones que guiaron su progreso. Luego, ponemos esta tesis en contexto con respecto a las tendencias actuales en el campo de las FPGAs y monitorización de redes, también comentamos el nuevo paradigma de FPGAs conectadas directamente a la red. Teniendo en cuenta estos aspectos, formulamos los objetivos de esta tesis doctoral. Finalmente, delineamos la organización de este documento y presentamos sucintamente el contenido del resto de los capítulos.

2.1 Motivación de esta tesis

Las actividades diarias se han vuelto cada vez más dependientes de las redes de ordenadores: Internet, aplicaciones móviles, comercio electrónico, aplicaciones en la nube, entre otros ejemplos. En este sentido, el uso generalizado de las redes de ordenadores tiene implicaciones tanto a nivel de acceso y nivel de servidores, por ejemplo, cada vez hay más necesidad de redes de ordenadores más rápidas y fiables. En el nivel de acceso, crea la necesidad del desarrollo de tecnologías de acceso más rápidas, por ejemplo, redes pasivas ópticas de 10 Gbit/s o la quinta generación de redes móviles (5G). Mientras que, en el lado del servidor ha causado un incremento exponencial en el tráfico que procesan los centros de datos. Desde este punto de vista, los autores en el artículo [1] discuten que este incremento exponencial no podría haber sido posible sin un incremento exponencial

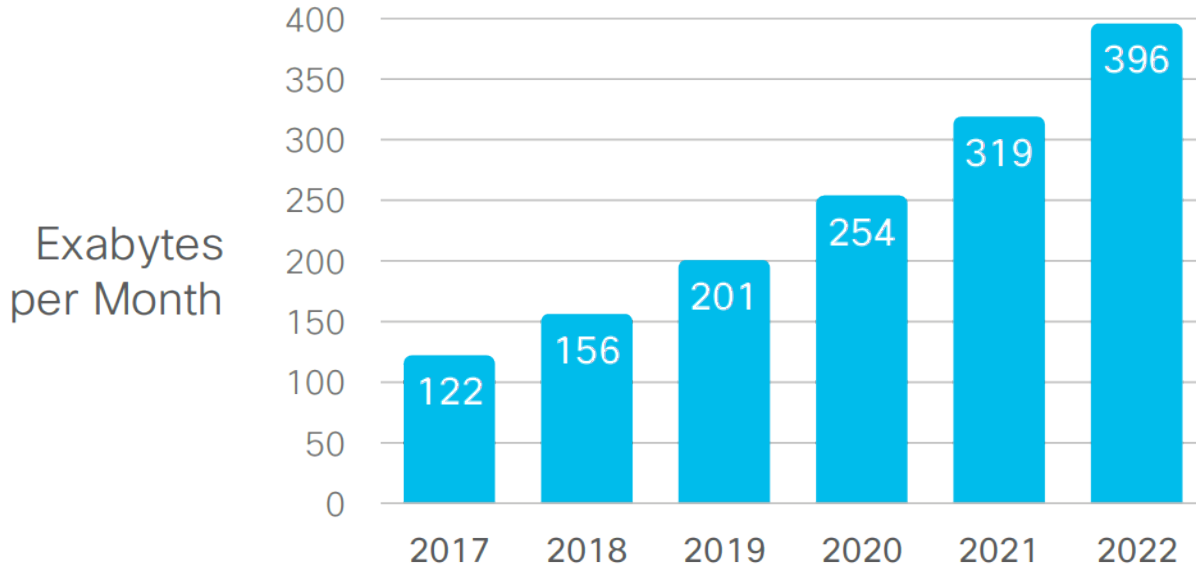


Figure 2.1: Tasa compuesta de crecimiento anual de tráfico IP. Fuente: [2].

en el ecosistema de los ordenadores: a nivel de circuito integrado, a nivel de sistema y en particular a nivel de adopción en la comunidad. Ellos también discuten que esos factores se realimentan para mantener el crecimiento a un ritmo exponencial. En este contexto, Cisco [2] predice que el tráfico IP crecerá exponencialmente durante el periodo 2017 a 2022, vea la figura 2.1. Más aún, en el artículo [3] se exploran los diferentes factores que impactan en el crecimiento continuo en Internet, por ejemplo, la figura 2.2 muestra el crecimiento exponencial en la demanda de banda ancha en el Reino Unido en un periodo de diez años medido durante la hora punta. Esta figura confirma la ley de Nielsen [4] del ancho de banda de Internet, que es una ley empírica basada en observaciones la cual afirma que *"La velocidad de conexión de un usuario crece un 50 % cada año"*.

Se espera que la velocidad de conexión de las redes de ordenadores continuará creciendo para equilibrar el crecimiento del tráfico de red, por consiguiente, el rendimiento de los equipos de cómputos debe equiparar este crecimiento. De esta manera, las redes de ordenadores se han convertido en una infraestructura donde el malfuncionamiento no se puede tolerar. Por lo tanto, es primordial garantizar la calidad de los enlaces de red para asegurar un funcionamiento adecuado de los sistemas informáticos. Como resultado, la monitorización de las redes de ordenadores es una parte esencial para asegurar su correcto funcionamiento y esta tarea es más necesaria que nunca. No obstante, la monitorización de las redes de ordenadores se convierte en una tarea compleja y costosa a tales velocidades; con el escalado de Dennard muerto (figura 3.3), los enfoques tradicionales basados en *software* tienen dificultades al monitorizar redes de varios gigabits por segundo. Últimamente, los circuitos específicos para abordar la monitorización de las redes de ordenadores a tal velocidad han ganado tracción para superar las limitaciones

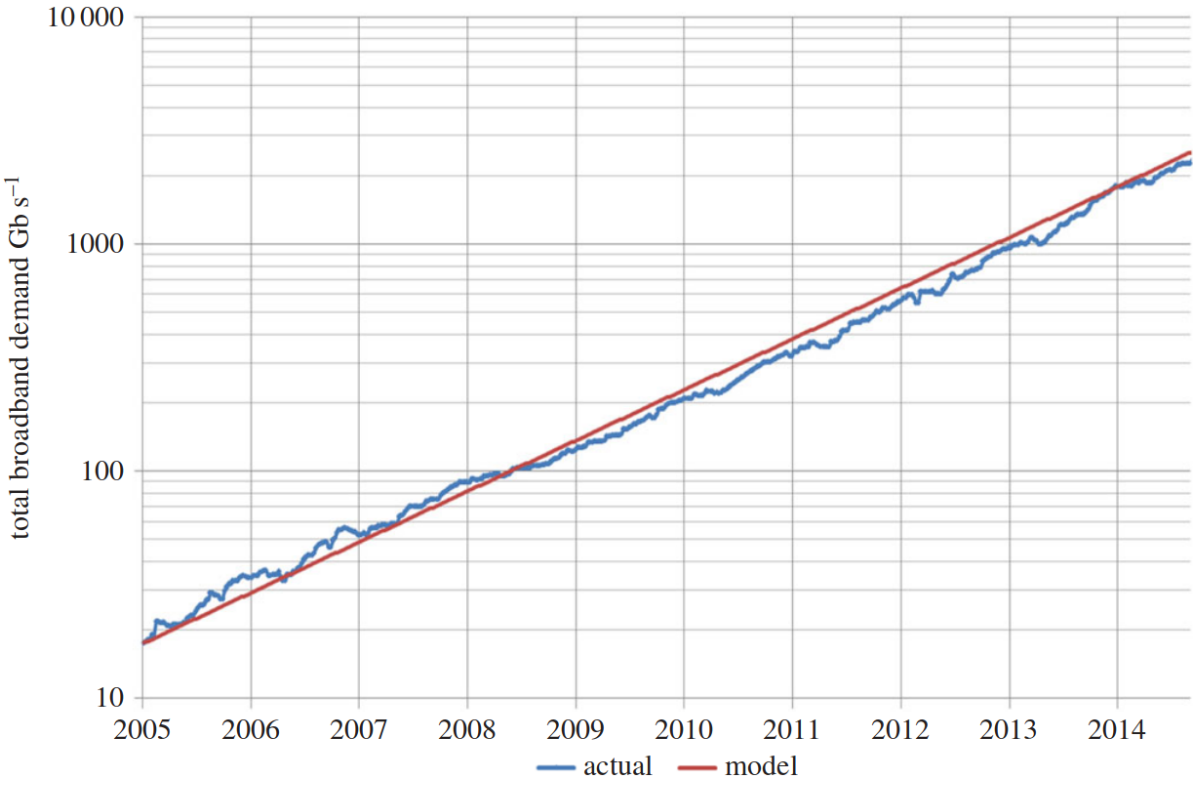


Figure 2.2: Demanda total de ancho de banda para un proveedor de Internet en el Reino Unido, medido en un periodo de diez años en la hora punta. Fuente: [3].

del *hardware* que se encuentra en las tiendas.

Las FPGAs han demostrado la capacidad de llevar a cabo una gran variedad de tareas de monitorización en el contexto de redes de ordenadores. En 2005 Taylor *et al.* [5] incluyó una implementación FPGA en su estudio de clasificación de paquetes de red. Por otro lado, en el artículo publicado en 2010 [6] los autores hacen una evaluación completa y minuciosa de las diferentes aproximaciones con FPGA en el contexto de seguridad en redes de ordenadores, ellos afirman que para las aplicaciones evaluadas la implementación basada en FPGA sobrepasa el desempeño de la implementación *software*. Además, en 2018 la empresa Arista [7] publicó un reporte con las cuatro tendencias en el uso de FPGA en el contexto de las redes de ordenadores — *Software Defined Network (SDN)*; aplicaciones de comercio automáticas sensitivas a latencia; captura de tráfico de red y marcado temporal; vídeo bajo demanda. Por lo tanto, las FPGAs han surgido como una tecnología ubicua no solo para tareas de monitorización de redes, sino que también para otras tareas relacionadas con las redes de ordenadores, por ejemplo, *switching* [8]. Cuando comparamos la capacidad de una FPGA con un ASIC, la FPGA provee más flexibilidad en términos de programación y menos costes recurrentes de ingeniería, mientras que mantiene un alto nivel de paralelismo. Basados en estas consideraciones, las FPGAs se posicionan como una de las soluciones más ventajosas

y económicas para solventar problemas complejos relacionados con la monitorización de redes de ordenadores. Además, NetFPGA ha sido una de las fuerzas impulsoras de las FPGAs en el contexto de redes de ordenadores. La iniciativa NetFPGA [9–11] es un proyecto de código abierto, tanto *hardware* como *software*. Empezó en 2006 como un proyecto de enseñanza en la universidad de Stanford y rápidamente se convirtió en una plataforma de prototipado tanto para proyectos de investigación como proyectos industriales.

Por otro lado, es conocido que el tiempo de desarrollo en FPGA es largo y complejo. Por muchos años los lenguajes de descripción de *hardware* (HDL por sus siglas en inglés) han dominado como describir *hardware* para diseños de FPGA. Sin embargo, en la última década los lenguajes de síntesis alto nivel (HLS por sus siglas en inglés) han demostrado suficiente madurez y calidad de resultados, es por ello que han ganado una parte importante en la fase de desarrollo en el mundo de las FPGAs [12]. Sucintamente, las herramientas HLS permiten describir *hardware* utilizando lenguajes de alto nivel, por ejemplo, C/C++ (introduciendo un mayor nivel de abstracción cuando se comparan con la metodología de nivel de transferencia de registro). En esta línea, en el artículo [13] los autores presentan una investigación exhaustiva de las herramientas HLS, en dicho artículo se discute que, aunque la calidad de resultados de las herramientas HLS no es tan buena como implementaciones personalizadas de RTL, la productividad es hasta cuatro veces mayor. En 2011 Cornu *et al.* [14] presentaron implementaciones eficientes de aceleradores de *hardware* desarrolladas en HLS que conseguían mayor desempeño que la implementación RTL. Forconesi *et al.* [15] presentaron una evaluación de Vivado-HLS (la herramienta comercial de Xilinx), ellos implementaron una aplicación de monitorización de flujos. En dicho artículo afirman que lograron una reducción de un orden de magnitud en el tiempo de desarrollo comparado con la metodología tradicional. Es más, una de las claves del éxito de las herramientas de HLS, aparte del hecho de que los principales lenguajes de programación son C y C++, es la habilidad de explorar el espacio de diseño mucho más rápido y generalmente sin la necesidad de cambiar el código fuente. A demás, en el último lustro la industria y la comunidad investigadora han introducido varias herramientas basadas en lenguajes de alto nivel para procesado de paquetes en el contexto de diseños FPGA [16–21]. Estos factores han alimentado el uso de herramientas HLS en el ámbito de redes de ordenadores. Por consiguiente, los lenguajes de síntesis de alto nivel son un tema de investigación muy activo.

Las FPGA no solo han sido usadas para tareas de redes de ordenadores, investigadores también las usan como nodo de computo en entornos de computación distribuidos [22–24]. Cuando comparamos a las FPGAs con unidades de procesamiento gráfico de propósito general (GP-GPU por sus siglas en inglés), las FPGAs proveen menos operaciones de coma flotante por segundo (FLOPS por sus siglas en inglés), sin embargo,

las FPGAs son más flexibles para implementar algoritmos irregulares. Lo que es más, la latencia de comunicación es mucho menor debido a que las interfaces están directamente conectadas al área de lógica programable [25]. A la luz de estas características, en los últimos cinco años ha habido esfuerzos para moverse del paradigma de FPGA conectadas a una CPU a un paradigma donde las FPGAs están conectadas directamente a la red [26–29]. Este fenómeno es consecuencia de la creciente heterogeneidad de las FPGAs. Fundamentalmente, la idea es desconectar la FPGA de una máquina anfitriona y conectarla directamente a la red. Con esto, se busca incrementar la eficiencia total del sistema y reducir el sobrecoste de comunicación. Este fenómeno no habría sido posible sin una implementación de TCP/IP a 10 Gbit/s que se hizo de acceso libre en 2015 [30]. Los esfuerzos en la comunidad han estado centrados en mejorar el nivel de abstracción de este paradigma. Es más, una de las líneas actuales de investigación está centrada en mover el procesamiento de los datos más cerca de la fuente de los mismos para reducir el sobrecoste de su movimiento [31]. Sin embargo, ha habido muy poco esfuerzo en aumentar la velocidad de comunicación en la infraestructura subyacente para adaptar el paradigma a las necesidades actuales. Por ejemplo, las redes neuronales convolucionales requieren un ancho de banda enorme en la comunicación de las capas convolucionales.

2.2 Contexto de esta tesis

Como se menciona anteriormente, las soluciones de *hardware* que se encuentra en la tienda (más su *software* asociado) tienen dificultades para mantener el ritmo a la que la velocidad de los enlaces de red ha evolucionado. Es por esto, que las FPGAs han surgido como una tecnología ubicua para abordar algunas de las tareas más complejas. Sin embargo, el desarrollo de *hardware* usando FPGA es considerado una tarea en la cual se necesitan muchas habilidades. No obstante, los diseños en FPGA están evolucionando hacia los lenguajes de alto nivel, estos no sólo reducen el tiempo de desarrollo, sino que también democratizan el uso de las FPGAs debido a una metodología de diseño mucho más simple. Por otro lado, como toda tecnología, las FPGAs tienen sus limitaciones, por ejemplo, requerimiento de una gran cantidad de memoria con un acceso de baja latencia. Sin embargo, las nuevas familias de FPGA incluyen memorias HBM como parte del silicio, esta nueva característica puede ayudar a mitigar esta limitación.

En este sentido, los conceptos descriptos previamente: monitorización de redes de ordenadores; FPGA en redes de ordenadores y síntesis de alto nivel han llamado la atención de la comunidad científica durante los últimos años. Notablemente, la mayoría de la investigación en estas líneas es contemporánea al periodo de esta tesis. La figura 2.3 muestra el número de referencias que se pueden encontrar en la web de la ciencia desde

el año 1990 a 2018 para las siguientes consultas:

- FPGA and network* NOT (“neural network*”)
- NetFPGA
- FPGA and “High Level Synthesis”

La figura muestra un incremento casi lineal de la cantidad de resultados en monitorización con FPGAs en el transcurso de los años, desde 1999. Este hecho confirma que las FPGAs han ganado parte del interés académico en este ámbito, por su virtud de aplicabilidad infinita. A pesar de esto, la mayoría de la investigación ha estado centrada en seguridad e inspección profunda de paquetes. El proyecto NetFPGA iluminó el camino en procesamiento de paquetes; esquemas de *switching* y dispositivos de monitorización. Antichi *et al.* [32] fueron algunos de los pioneros en monitorización pasiva usando la tarjeta NetFPGA. En dicho trabajo, presentan un sistema basado en la cooperación de una NetFPGA y un sistema anfitrión de propósito general. Como continuación de esta línea investigativa, en el artículo [33] presentan OSNT: un sistema de código abierto para la evaluación de redes de ordenadores, que incluye generación de paquetes como así también capacidad para captura de tráfico. De la misma forma, Oeldemann *et al.* [34] presentan un sistema para evaluar redes, muy similar a OSNT, pero con pequeñas mejoras en la reproducción de trazas y en la programación *software* a través de su interfaz de programación. Es más, Puš *et al.* en 2015 presentaron una tarjeta hecha a medida usando

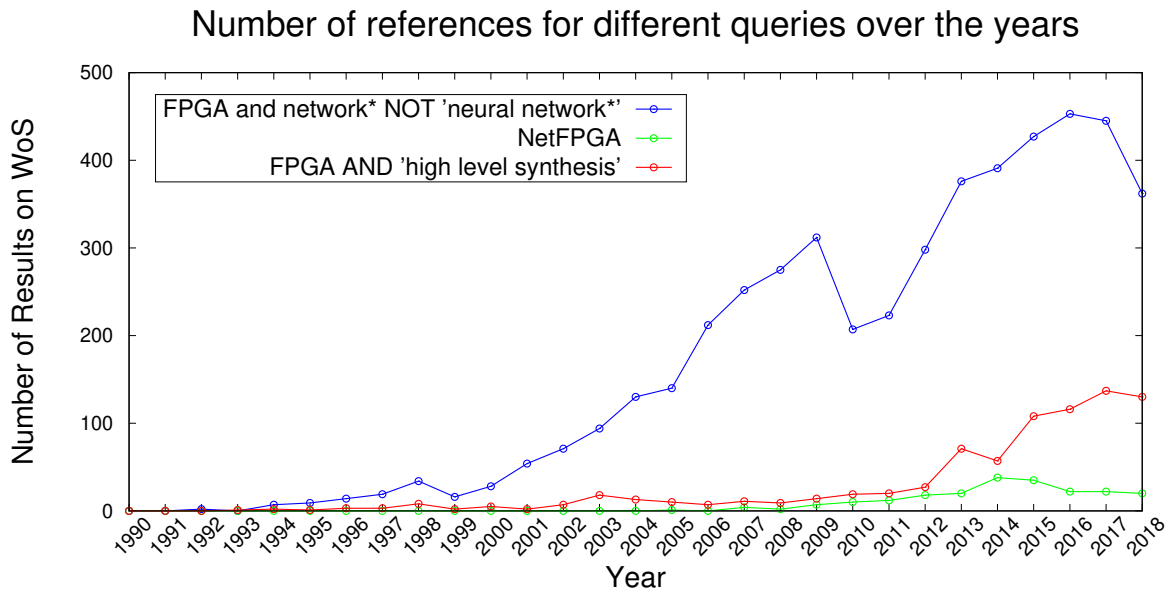


Figure 2.3: Progresión del número de trabajos publicados disponibles en la web de la ciencia por cada una de las consultas, desde el año 1990 a 2018

una FPGA capaz de manejar enlaces de 100 Gigabit Ethernet, la FPGA distribuye el tráfico entre los diferentes núcleos de la máquina anfitriona para ser luego procesado.

Con relación a HLS, la investigación se ha centrado en la aceleración de algoritmos, especialmente en el ámbito de aceleradores de aprendizaje automático. A pesar de esto, ha habido esfuerzos hacia sistemas de monitorización de redes usando HLS [15, 35]. La introducción de P4 [36], un lenguaje de dominio específico optimizado alrededor de los datos, guio el camino hacia modelos de abstracción de alto nivel en el ámbito de procesado de paquetes de red [17–19, 37–39]. En 2017, Emu [40] fue introducido para promover el desarrollo rápido de funcionalidades de redes. Es más, Eran *et al.* [20] presentaron una biblioteca para procesado de paquetes de red con el objetivo de aumentar la reutilización de código, esta biblioteca está construida sobre HLS.

Asimismo, la creciente demanda por computación distribuida, desempeño y adaptabilidad de muchas aplicaciones distribuidas depende de una implementación eficiente del protocolo de comunicación TCP/IP. La implementación de un acelerador de dicho protocolo en FPGA abre interesantes oportunidades para explorar tarjetas de red programables, ya que estas ofrecen mayor flexibilidad que las tarjetas de red comerciales. Desafortunadamente, alcanzar 100 Gbit/s es un desafío complejo incluso usando tarjetas de red comerciales con aceleración TCP. De ahí que, el trabajo Sidler *et al.* [30] sentó las bases con una implementación completa del protocolo TCP/IP a 10 Gbit/s para FPGA. A partir de esto, la mayoría de los desarrollos de FPGA conectadas directamente a la red han sido construidos sobre dicho trabajo [26, 41].

Considerando todo lo anteriormente mencionado, la figura reconoce el incremento del interés en la comunidad científica en el uso de FPGA para tareas de redes de ordenadores, a su vez que también demuestra el crecimiento en los esfuerzos por el uso lenguajes de alto nivel. Por otro lado, el número de resultados de NetFPGA se ha quedado más o menos estable desde 2015, esto se puede deber a que la última versión de la plataforma data del año 2014.

2.3 Objetivos de esta tesis

En la sección anterior, presentamos las líneas de investigación abordadas en esta tesis. Por consiguiente, en esta sección describimos los objetivos principales de esta tesis. Establecemos tres objetivos principales. El primero profundiza en la capacidad de las FPGAs en el ámbito de la monitorización de redes de ordenadores, tanto en escenarios de monitorización activos como pasivos. El segundo objetivo busca la implementación de un sistema de transmisión de datos fiable, de alta velocidad y con baja latencia en el contexto de comunicación de nodos de FPGAs distribuidos, donde las FPGAs están

desconectadas de máquinas anfitrionas y conectadas directamente a la red. Finalmente, el tercer objetivo evalúa la aplicabilidad de HLS en la metodología de diseño de FPGA para aplicaciones de redes de ordenadores, este objetivo está completamente empotrado en los dos anteriores.

Los siguientes ítems resumen los sub objetivos de esta tesis:

- **Explorar las capacidades de las FPGAs en monitorización activa de redes de ordenadores**: las FPGA son conocidas por su alto grado de paralelismo y su determinismo. Queremos sacar provecho de estas características para llevar a cabo mediciones activas. En particular, queremos contrastar exactitud de las FPGA contra las soluciones de *software* usando la técnica de los trenes de paquetes.

- **Explorar las capacidades de las FPGAs en monitorización pasiva de redes de ordenadores**: en este ámbito queremos aprovechar la heterogeneidad de las FPGA para reducir el tráfico de red inteligentemente sin perder información relevante en el camino. Aquí exploramos como recortar paquetes cifrados, y como remover paquetes duplicados. La idea es ayudar a las herramientas de *software* tradicional para no procesar datos innecesariamente.

- **Explorar las capacidades de las FPGAs para implementar el protocolo TCP/IP completo**: el nuevo paradigma de FPGA desconectadas de máquinas anfitrionas necesita de una implementación eficiente del protocolo de comunicación. TCP ha sido el estándar *de facto* para transmisión de datos fiables, a pesar de eso, es conocido que dicho protocolo es muy demandante debido a la gran cantidad de memoria requerida para almacenar el estado las cada una de las conexiones. Nosotros queremos llevar las capacidades de las FPGA al límite y proveer a la comunidad con una implementación FPGA de código libre del protocolo TCP/IP a 100 Gbit/s con muy baja latencia, a partir de la cual cualquier persona pueda construir su aplicación.

- **Evaluar la aplicabilidad de los lenguajes de alto nivel**: esta tesis presenta una oportunidad única para evaluar código HLS (C/C++) en el contexto de una gran cantidad de aplicaciones de redes. Entonces, trataremos de aplicar HLS la mayor cantidad de veces posibles en nuestros desarrollos. En particular, usaremos Vivado-HLS la herramienta comercial de Xilinx.

Esta tesis tiene un gran componente experimental. Es por ello que cada uno de estos objetivos han sido verificados con implementaciones en la forma de pruebas de concepto, y en algunos casos proponemos más de una solución para el mismo problema.

2.4 Estructura de la tesis

Para finalizar la introducción de esta tesis, esta sección resume el resto del documento. La tesis está dividida en cuatro partes, ***Parte I Introduction****: el capítulo 3* introduce el entorno técnico de las diferentes tecnologías utilizadas en el desarrollo de esta tesis. También provee sucintamente la historia de los circuitos integrados, redes de ordenadores y FPGAs. A demás, presentamos algunos conceptos básicos acerca de monitorización de redes de ordenadores. La idea principal de este capítulo es poner a lector en contexto de las diferentes tecnologías usadas en esta tesis. Finalmente, mostramos todas las tarjetas de desarrollo usadas durante la tesis.

Parte II Network Monitoring: en esta parte exploramos las capacidades de las FPGAs para llevar a cabo tareas de monitorización de redes. En particular, el *capítulo 4* se centra en monitorización activa, donde aprovechamos el determinismo de las FPGAs para implementar la técnica de trenes de paquetes con el objetivo de verificar la calidad de servicio de un dispositivo bajo pruebas. La escalabilidad de la técnica previamente mencionada, así como su implementación FPGA ha sido demostrada verificando su funcionamiento en diferentes velocidades de enlace — 1, 10 y 100 Gbit/s. Luego, el *capítulo 5* tiene el objetivo de proveer implementaciones que se conecten antes de las herramientas tradicionales de *software* para reducir el tráfico que llegan a estas, de esta manera se pretende ayudar a las herramientas tradicionales a cumplir con su objetivo. En este capítulo exploramos alternativas para hacer la reducción del tráfico sin perder información relevante. A demás, presentamos dos implementaciones donde los paquetes cifrados son recortados y los paquetes duplicados son removidos. En este sentido, exploramos la heterogeneidad de las FPGAs.

En la ***Parte III Offload Tasks***, exploramos la arquitectura de las FPGA para implementar tareas muy demandantes, pero en este caso el objetivo es descargar parte o la totalidad del protocolo TCP/IP a una FPGA. Primero en el *capítulo 6* exploramos diferentes circuitos para descargar la computación de la suma de comprobación de 16-bit en complemento a uno, que es usado en todas las cabeceras de IP versión 4. Luego en el *capítulo 7* presentamos Limago: una implementación FPGA del protocolo TCP/IP que funciona en redes de 100 Gigabit Ethernet, el cual está basado en un trabajo previo. Limago multiplica por diez la velocidad de comunicación cuando se compara con su predecesor, al mismo tiempo mantiene la escalabilidad en termino de conexiones concurrentes. Para lograr este objetivo, usamos HLS no solo para acelerar el proceso de desarrollo, sino también para ganar más flexibilidad. Limago contribuye con un paso hacia adelante en la diseminación de FPGA en entornos distribuidos.

Finalmente, en la ***Parte IV Conclusions***, los capítulos *8 y 9* en inglés y español respectivamente resumen las conclusiones y proveen algunas observaciones extras.

Adicionalmente, presentamos las contribuciones de esta tesis a la comunidad científica, también incluimos una discusión de cuando tiene sentido utilizar síntesis de alto nivel y por último las direcciones del trabajo futuro en esta área de investigación.

CHAPTER 3

TECHNICAL BACKGROUND

In this chapter we set in context the different technologies that clash in this Thesis. In Section 3.1, we start with a brief history of computers as well as computer networks, then we present FPGAs. Secondly, in Section 3.2, we motivate the need of network monitoring and we present both active and passive alternatives. After that we move to the FPGA arena, Section 3.3 introduces the design tools: Vivado and Vivado-HLS. In section 3.4 we shed light on how FPGAs are able to handle multi-gigabit-per-second networks. Section 3.5 presents the standard AXI4 specification, which is the underlying communication infrastructure of all of our designs. Finally, we show the different FPGA platforms used in this Thesis (Sections 3.6 to 3.8).

3.1 Background

The first computers were enormous mechanical monsters, power hungry, complex to use and prone to failures as well as costly. Nonetheless, when the first transistor was invented (1947) a new era begun. A decade later, using a few transistors the first integrated circuit was create (1958) and the silicon revolution started. More powerful and easy-to-use computer were built. What is more, based on empirical observations, Moore's Law state that the amount of transistors within an integrated circuit doubles every two years [42]. Years after, the law was revisited and the time span was updated to eighteen months. In any case, the exponential growth in the number of transistor has been proven and stays valid to these days [43, 44]. In this context, Figure 3.1 plots

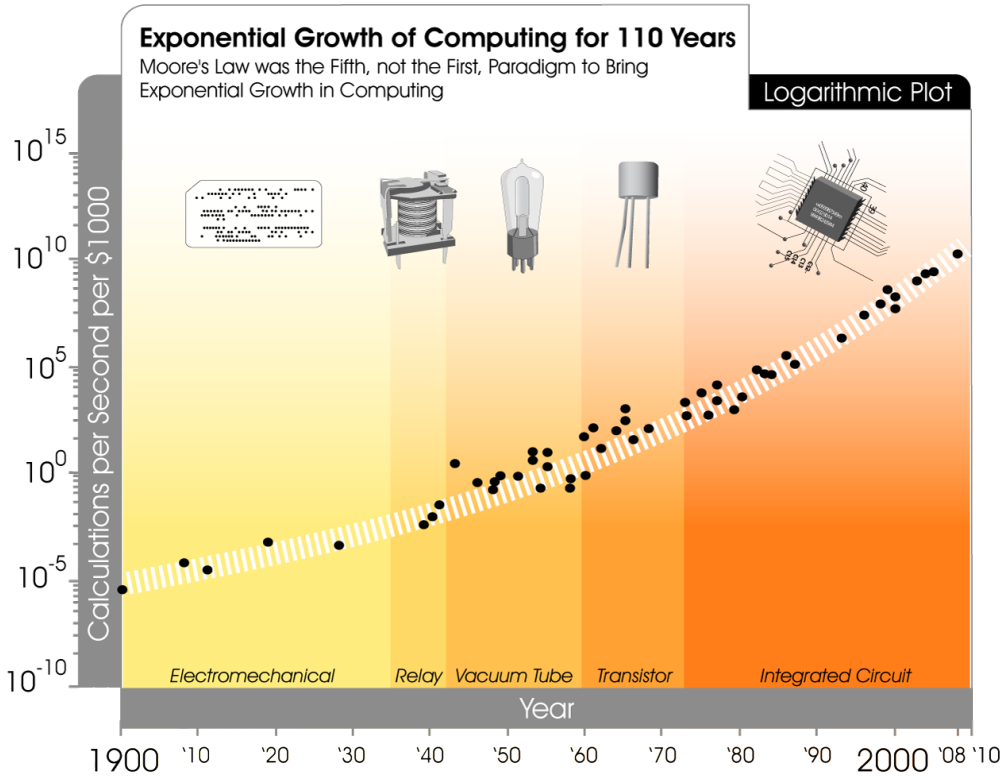


Figure 3.1: Logarithmic scale of computation per second per thousand dollars since 1900, spans five families of technologies. The computation per second doubles every 1.3 years approximately. Source Kurzweilai. [1]

the exponential growth in calculations per second per $ 1000 in the span of 110 years, the invention of the transistor was a breakthrough which allowed cheaper and more powerful computation systems. However, the figure also shows that regardless of the technology the exponential growth in the computation power has been a constant.

Nonetheless, computers were isolate from each other, therefore large storage systems were used to save data and some mechanism to move data were created. Consequently, the need for a communication system across computers arose. Therefore, when computers could communicate to each other, more complex systems were built. At the beginning, the computer networks were only accessible to the government, universities and big companies. However, in the 1990s the democratization of the Internet begun. A few years later, another empirical law originated, Nielsen's Law [4], which states that *"Users' bandwidth grows by 50% per year"*, a 10% slower than Moore's Law. Figure 3.2 plots the end-user connection speed between 1983 and 2018, the actual data fit the Nielsen's law estimation. Shortly afterwards, computer networks evolved into mainstream, and thus monitoring them became key in order to control the network healthiness. Not only Internet Service Providers (ISPs) are interested on keeping the Quality of Service (QoS)

[1] Source: "The Singularity Is Near: When Humans Transcend Biology". Author Ray Kurzweil

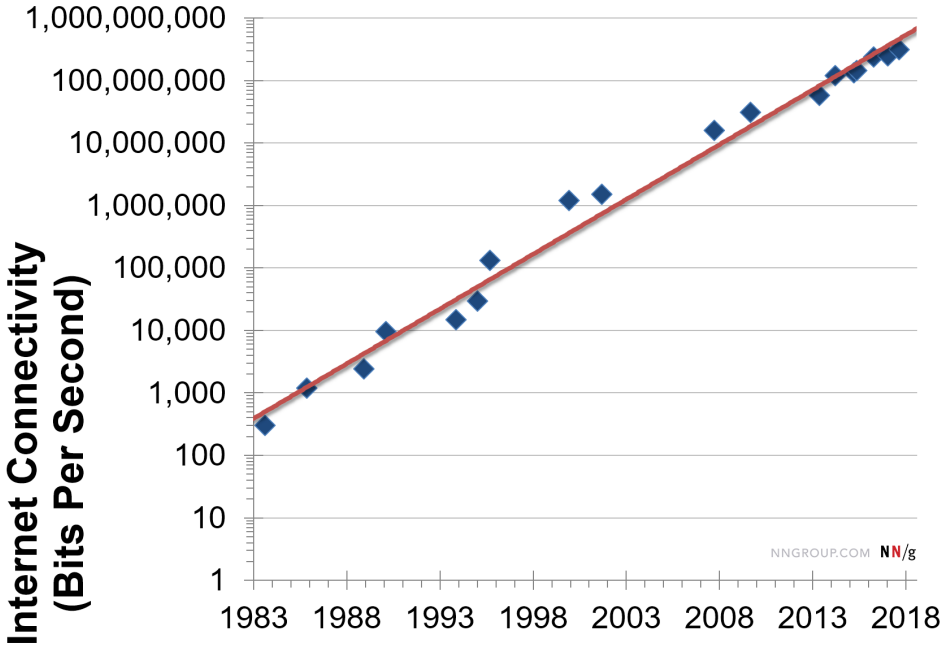


Figure 3.2: Logarithmic scale of access speeds of an end-user over the years. The empirical model fits very well the actual data. [2]

of a network, but also, it has become an active research topic. The books [45, 46] provide a wider explanation of the history of computers and networks. In this context, Edholm's law of bandwidth [47] splits the access technologies in three categories — wireline, nomadic and wireless — and states *"the three telecommunications categories march almost in lock step: their data rates increase on similar exponential curves, the slower rates trailing the faster ones by a predictable time lag"*. Extrapolating the results presented such work, the rate of nomadic and wireless technologies will converge in 2030.

Even though, Moore's Law still alive, Jerry Wu *et al.* [48] evaluate the current limitation of the Moore's Law and how the amount of transistor could continue to growth in the future. The computing performance of a processor does not only depend upon the number of transistors. Another important factor in this matter is the frequency. For many years, the vendors were able to increase the operating frequency. To do so, the voltage operation was lowered so as to reduce the heat dissipation. This span of time was called Dennard scaling [49, 50]. In the early 2000s, the increase in frequency stalled mainly due to heat dissipation. Therefore, new processors with multi and many core were built to keep increasing the computing performance. Lately, the overhead of intra core communication is slowing down on the computing performance of multipurpose processors. Figure 3.3 depicts the performance of a processor over the years, the performance slowed down gradually, and, it appears that since 2015 the performance growth has stalled. Yet, some

[2] Source: `https://www.nngroup.com/articles/law-of-bandwidth/`

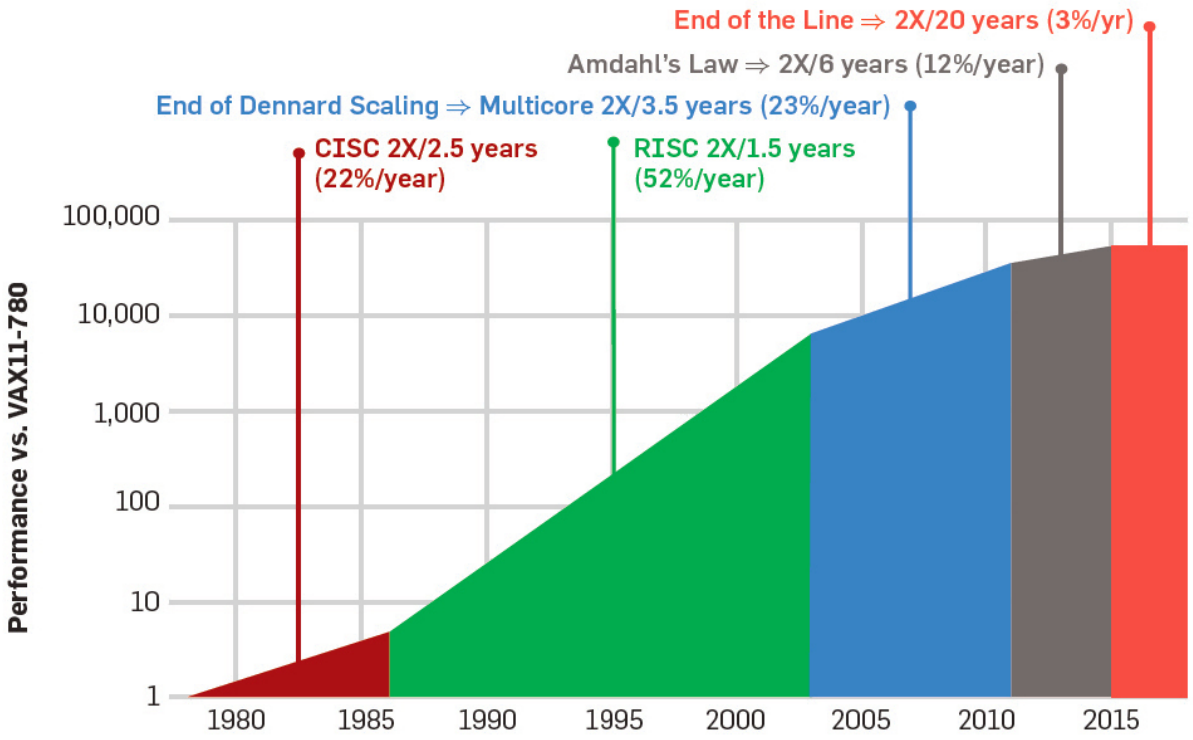


Figure 3.3: Growth in processor performance over 40 years, SPEC integer benchmarks. Source: [51].

authors argue that to solve challenging problems the only path is designing application specific circuits [51, 52].

On the other hand, in 1984 Xilinx introduced the first Field Programmable Gate Array (FPGA) as an alternative to Application Specific Integrated Circuit (ASIC). The FPGAs are more cost-effective because of their reduced non-recurrent engineering costs in terms of development time. Nevertheless, FPGA consumes more power to perform the same task. An FPGA is an ASIC but with a highly degree of programmability. What is more, the design can change over the time, thus the same chip can be used for different applications. Fundamentally, the architecture of an FPGA is made of an array of programmable logic block and a matrix of interconnection to communicate those blocks. This architecture provides scalability improvement over the Programmable Array Logic (PAL) its predecessor. The first commercial FPGA had only sixty-four logic block, each with two three-input LUT and one register, and it has a huge die (silicon area) when compare to the state-of-the-art microprocessor at that time. In those days, the designs were done manually given the little complexity. However, following Moore's Law the amount of transistors growths, thus the amount of programmable logic growths at an exponential pace, and therefore the complexity. As the complexity of FPGAs growth, the need for abstraction languages and design tools arose. Therefore, Register Transfer Level (RTL) was established as the main programming model for FPGA, being Very

High Speed Integrated Circuit Hardware Description Language (VHDL) and Verilog the two most popular languages. Afterwards, FPGAs started to incorporate more complex pieces of silicon to address a wider variety of problems. For instance, on-chip memory, specialized multiplier accumulators, even microprocessors. Trimberger [53] describes the evolution of FPGAs in three phases: 1) Age of Invention (1984-1991); 2) Age of Expansion (1992-1999) and 3) Age of Accumulation (2000-2007). He also states that FPGAs have been the driving force of the network equipment since 2000's, because their capability of changing functionality in the field, consequently, FPGAs have evolved to adapt to network tasks. Nowadays, FPGAs can be found in a wide range of applications, but mostly in high-performance systems and low-latency applications. What is more, FPGAs are moving towards a more heterogeneous architecture with different subfamilies to tackle effectively different type of problems [54]. Furthermore, one of the key aspects of FPGA when compared to processors is its huge parallelism.

3.2 Network Monitoring

Computer networks have become a critical infrastructure where malfunction cannot be tolerated. Therefore, monitoring the healthiness of such networks is key for providing reliable network and complying Service Level Agreements. The infrastructure managers use the results of the monitoring tools in order to obtain insight of the status of the network at a given time. And, based on such results they can take actions to solve problems. Lee *et al.* [55] provides a survey of different monitoring techniques. In particular, we can distinguish two types of monitoring, active and passive. The main difference resides in the nature of the traffic, which can be artificial or real. In what follows we present both alternatives with their pros and cons.

3.2.1 Passive Monitoring

Passive monitoring captures the real traffic in a certain point in the network. After that, the traffic is analyzed in order to estimate the network behavior. The passive probes can collect raw traffic or summaries coming from specialized systems, being NetFlow one of the most popular. What is more, the traffic analysis can be done a) in real-time to provide proactive action in the network, or b) on-demand, for instance, when a problem occurs the analyst can recover the traffic and perform the analysis. This type of monitoring leads to reactive actions into the network, for instance, actions must be taken after a problem occurs. Large and fast storages systems are needed to save the captured traffic for future analysis [56, 57]. Needless to say, real-time analysis can be very demanding for today's networks speeds.

3.2.2 Active Monitoring

Active measurement demands to inject traffic into a network with the goal of measuring its characteristics. Such approach allows to obtain reliable statistics and determine the Key Performance Indicators (KPI) of a segment of the network. This type of measurements can be applied to any kind of network infrastructure or equipment. This approach allows to measure, throughput (link or path capacity), latency — Round-Trip delay Time (RTT) or One-Way Delay (OWD) (time synchronization is needed) —, packet loss ratio and jitter. The injected traffic can be generated in such a way that triggers corner cases that in a normal scenario would be extremely difficult to catch. However, this approach is highly intrusive, because additional traffic is injected into the network, hence the normal behavior of the network is perturbed. This kind of measurement is often used to validate a path or network equipment. What is more, the interference introduced to the real traffic depends upon the frequency and length of the synthetic traffic injected.

3.3 Tools and Design Flow

In this section we present the different FPGA design tools we have used in the development of this thesis. We have used Xilinx's FPGAs, therefore, we only refer to Xilinx tools. Sometimes these types of tools are called Electronic Design Automation (EDA) tools.

3.3.1 Vivado

It was introduced in 2012 to replace ISE, the previous EDA tool. Vivado is the Xilinx's EDA to translate digital designs from RTL designs abstraction level to a bitstream — a set of bits that configures the FPGA to implement a digital circuit. Vivado allows the user to synthesize (maps the operation to the programmable logic), simulate and perform timing analysis. In order to obtain the final bitstream the implementation must be performed, the resources are placed and route in the target FPGA. Apart from the traditional RTL design methodology, Vivado incorporates the IP integrator tool, which is a graphical tool to connect the IP-Cores at interface level, this tools helps with the integration of large designs. It simplifies traditional design methodology and reduce the probability of errors in the interconnection of modules, the connection of a complex interface such as AXI4-Full is reduced to draw a line between two modules — the tool automatically connects the input and outputs properly. Figure 3.4 shows how IP integrator looks like, this figure is a piece of Limago (chapter 7).

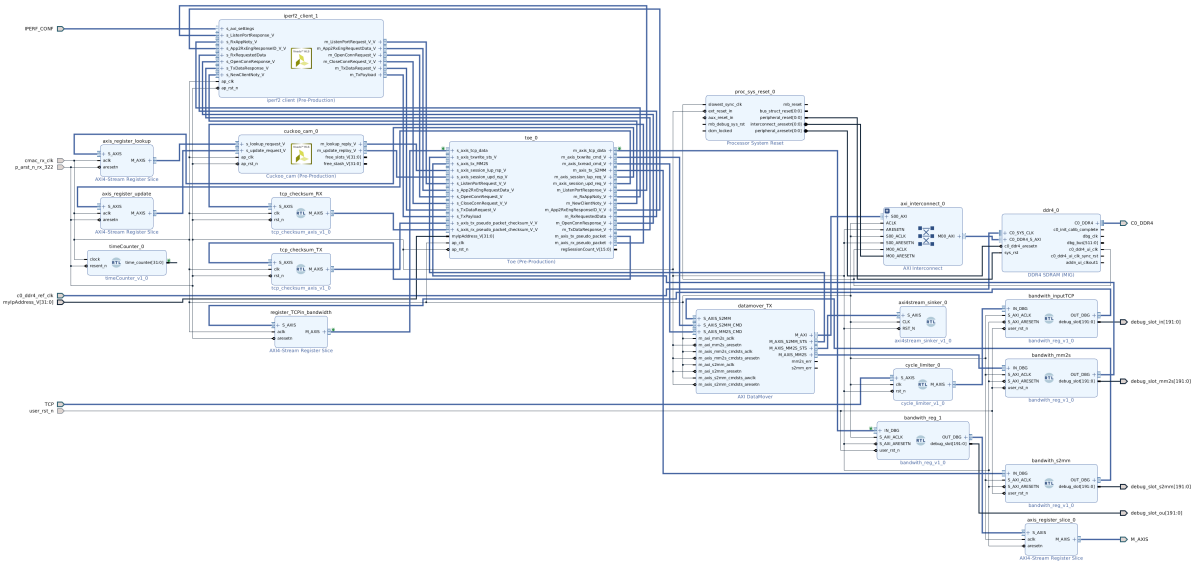


Figure 3.4: Vivado IPI, part of the block design of Limago.

3.3.2 High-Level Synthesis

There are a wide variety of High-Level Synthesis (HLS) tools, but in this thesis we have only used Vivado-HLS, which is the commercial tool from Xilinx Inc. Vivado-HLS aims at translating a behavioral C/C++ algorithm into a digital circuit that implements such behavior. In particular, Vivado-HLS provides an IP-Core as a result which can be integrated in any design flow of Vivado suite. Figure 3.5 summarizes the Vivado-HLS design flow. The tool uses the description of an algorithm in any of the supported languages, and after the C Synthesis RTL code is generated. At the same time, the user can provide testbenches to verify the algorithm functionality, in this step a C simulation can be carried out, or a RTL simulation. Noteworthy, the tool uses the same set of testbenches to run both C and RTL simulation. Once the user obtains the desired result the RTL code is packaged using a IP-XACT standard format, and it can be used in any other tool of Xilinx's environment. The C Synthesis uses the constraints/directives to translate the behavioral algorithm into an RTL design. Thus, the tool performs the following steps in order it fulfill its purpose:

1. Scheduling: determines which operation occurs during each clock cycle taking into account data dependencies, this process depends heavily upon user constrains.

2. Binding: maps each schedule operating onto the hardware resources available in the target device. Constrains allow a certain degree of flexibility on the mapping.

Figure 3.6 depicts an example of how Vivado-HLS schedules and binds a simple piece of code to a hardware implementation. What is more, Vivado-HLS generates a FSM to

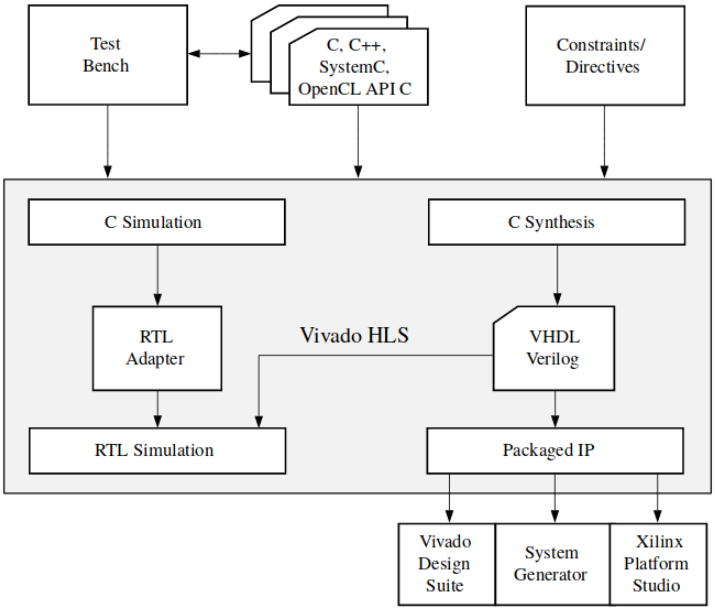


Figure 3.5: Vivado-HLS design flow. [3]

sequence the operation into the RTL design. The constraints allow the user to explore the space design without modifying the source code. Among the HLS benefits we can highlight: improve productivity; develop and verify at C level and quick space design exploration.

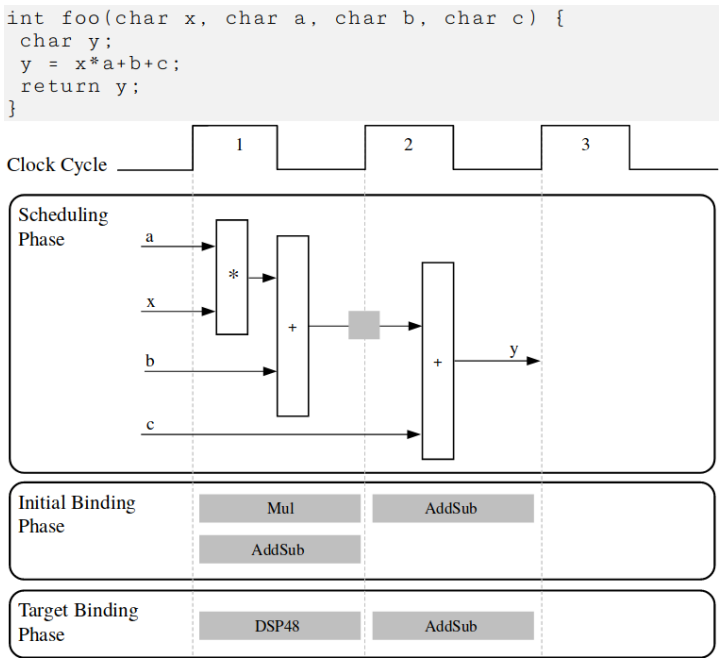


Figure 3.6: Vivado-HLS scheduling and binding example. [3]

[3] Source: https://www.xilinx.com/support/documentation/sw_manuals/xilinx2019_1/ug902-vivado-high-level-synthesis.pdf

3.4 FPGA for Networking

In this section we shed light on how FPGAs are able to handle multi-gigabit-per-second input/output pins. This feature, allows FPGAs to support a wide variety of Ethernet speeds as well as standards. Furthermore, we explain how FPGA translates the physical bits into frames that can be processed within the programmable logic.

3.4.1 Multi Gigabit Transceivers

It is well-known that the operating frequencies of FPGAs are much lower than ASICs, around one order of magnitude less. Therefore, to handle very high-speed inputs and outputs. FPGAs take advantages of Multi Gigabit Transceiver (MGT). MGTs are specialized hardware able to Serialize and Deserialize data at rate above 1 Gbit/s. On the one hand, in the input side this piece of hardware receives data at very high-speed in a serial fashion and parallelize it — using a Serial In Parallel Out (SIPO), in such a way that the operating frequency inside of the FPGA can be reduced. On the other hand, in the transmitting side, the FPGA provides data through a bus and the MGT serialize it to be outputted — using a Parallel in Serial Out (PISO). MGTs have been paramount to support multiple network speeds as well as different standards. Currently, the state-of-the-art MGTs are able to reach up to 112 Gbit/s.

3.4.2 Interpreting the bits

Following the Open Systems Interconnection (OSI) layers: the physical layer described how the bits are encoded and transmitted over the medium. This is specified, for instance, in the RJ45, SFP+ and QSFP28 standards. Typically, 10 and 100 GbE interfaces incorporate a chip to decode the modulation and provides a chip to chip interface, for instance, 10G Attachment Unit Interface (XAUI) for 10 GbE and CAUI-4 (100 Gigabit Attachment Unit Interface four lanes) for 100 GbE. This interface is connected to the MGTs to perform the serialization and deserialization. After that, the FPGA implements the Media Access Control (MAC) which as output provides interpretable frames. Each frame is split into multiple $n-bit$ wide transactions, the width of the transaction depends on the Ethernet speed and can vary from 32-bit to 512-bit (1 and 100 GbE links). Up to 40 GbE Xilix provides soft IP-Cores to implement the MAC module, which means logic is needed to implement such functionality. However, to support 100 GbE Xilinx designed a harden IP-Core, which means the implementation is a piece of silicon and does not occupy logic resources when used. For instance, Figure 3.7 shows the different elements in the architecture of 10 Gigabit Ethernet. Usually, the MAC core in the FPGA interface the user logic with an AXI4-Stream interface (Section 3.5.2).

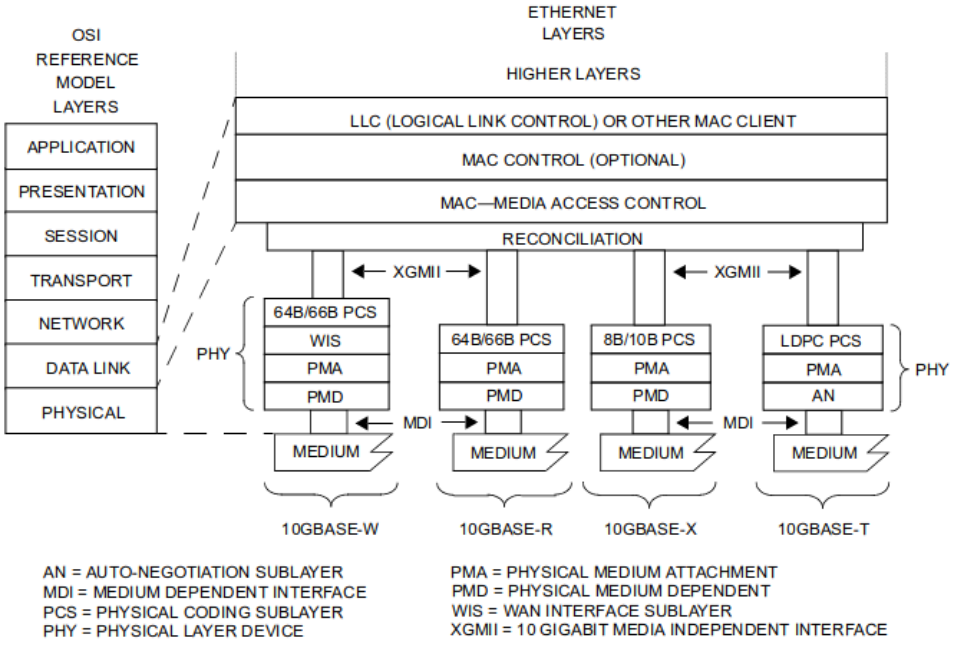


Figure 3.7: Architectural positioning of 10 Gigabit Ethernet. [4]

3.5 AXI4 Specification

Advanced eXtensible Interface 4 (AXI4) is part of the Advanced Microcontroller Bus Architecture (AMBA) interface specification from ARM [58, 59], which was introduced in 2010. It is an open standard, and it has become the *"de facto"* connection and management interface in today's logic designs. Xilinx introduced the AXI4 standard as part of the design methodology with the release of their System on a Chip (SoC) [60], since then every new FPGA family has supported it.

AXI4 is a point to point interface, there are always a master and slave. It has three flavors: two memory map oriented and one stream-oriented — no address is required. The memory mapped flavors have separated address/control and data phases. The support for unaligned transfers is given by the strobe bytes. One of the benefit AXI4 provides is a consistent signalization among the different flavors. In such a way, the user only needs to learn once the protocol and can apply it to any kind of design. Table 3.1 summarizes the characteristics of each AXI4 flavor and provides some examples of it applicability.

3.5.1 AXI4-Full and AXI4-Lite

AXI4-Full: For high-performance memory-mapped requirements, to achieve so, the concept of a burst is used, in which the address is provide once, and the access follows a sequential pattern, incremental or decremental, up to 256 transactions.

[4] Source: IEEE 802.3-2018 - IEEE Standard for Ethernet

AXI4-Lite: It is the lightweight version of AXI4-Full with no burst support. It is used for simple, low-throughput memory-mapped communication for instance, to and from control and status registers.

AXI4-Full and AXI4-Lite have five independent channels:

- Read Address Channel
- Write Address Channel
- Read Data Channel
- Write Data Channel
- Write Response Channel

The data transfer can occur in both directions between master and slave at the same time. The data transfer size range depending on the application. However, the limit is up to 256 transactions per burst. On the other, hand AXI4-Lite just support one data transfer per burst.

3.5.2 AXI4-Stream

The AXI4-Stream is used for high-speed streaming data where no address is involved. It has single channel for transmission of streaming data. It is similar to the write data channel of an AXI4-Full. The bursts are unlimited contrary to AXI4-Full. Since there is no addressing involved, the transactions follow a strict order, thus, cannot be reordered. This is the standard way to communicate packets within Xilinx FPGAs.

3.6 1 Gbit/s Platform

ZebBoard is a low cost board based on a Xilinx Zynq SoC (an XC7Z020-CLG484-1) device that encompasses in a single device a microcontroller based on a dual core ARM

	AXI4-Full	AXI4-Lite	AXI4-Stream
Dedicate for	high performance and memory mapped systems	register-style interfaces peripherals	non-address based
Burst support	up to 256	No	Unlimited
Data width	32 to 1,024-bit	32 or 64-bit	any number of bytes
Application	Embedded devices and memory mapped	Small footprint control logic	DSP, video, communication

Table 3.1: Summary of different AXI4 flavors.

Cortex-A9 (namely PS, Processing System) and an FPGA (namely PL, Programmable Logic). The board has plenty of input/output connectors, standing out an FPGA Mezzanine Card (FMC) connector that allows plugging complex peripherals to the system. Furthermore, an operating system such as Linux can be run in the PS, thus enabling complex applications to be developed. Besides, the PL was the state-of-the-art technology at that time (Xilinx 7-Series), which allows building complex hardware peripherals in the form of hardware modules (also known as IP-Cores, from Intellectual Property core). We attached two external Gigabit Ethernet interfaces through the FMC connector directly to the programmable logic. Figure 3.8 shows the board with the FMC daughter card.

3.7 10 Gbit/s Platform

NetFPGA is an open-source hardware and software project, developed by Stanford and Cambridge Universities in collaboration with Xilinx. It is intended for rapid prototyping of computer network devices. The NetFPGA-10G is based on Xilinx Virtex-5 FPGA (an XC5VTX240TFFG1759-2). It provides four SFP+ interfaces and has an 8X PCI Express Gen 1 interface to the host. Even though it can work standalone, it is typically attached to a host machine. Figure 3.9 shows the NetFPGA-10G board, the four interfaces the left hand side are the cage where the SFP+ connector is plugged and the next to them are Physical Layer (PHY) chips.

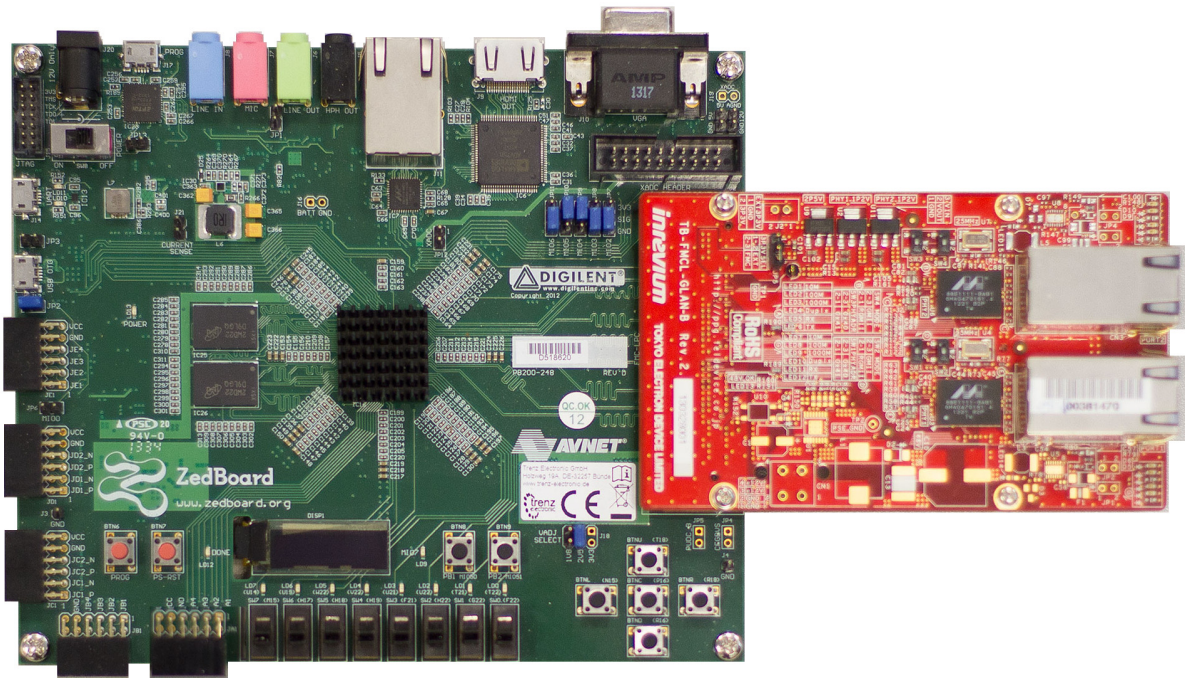


Figure 3.8: ZedBoard development board from Avnet.

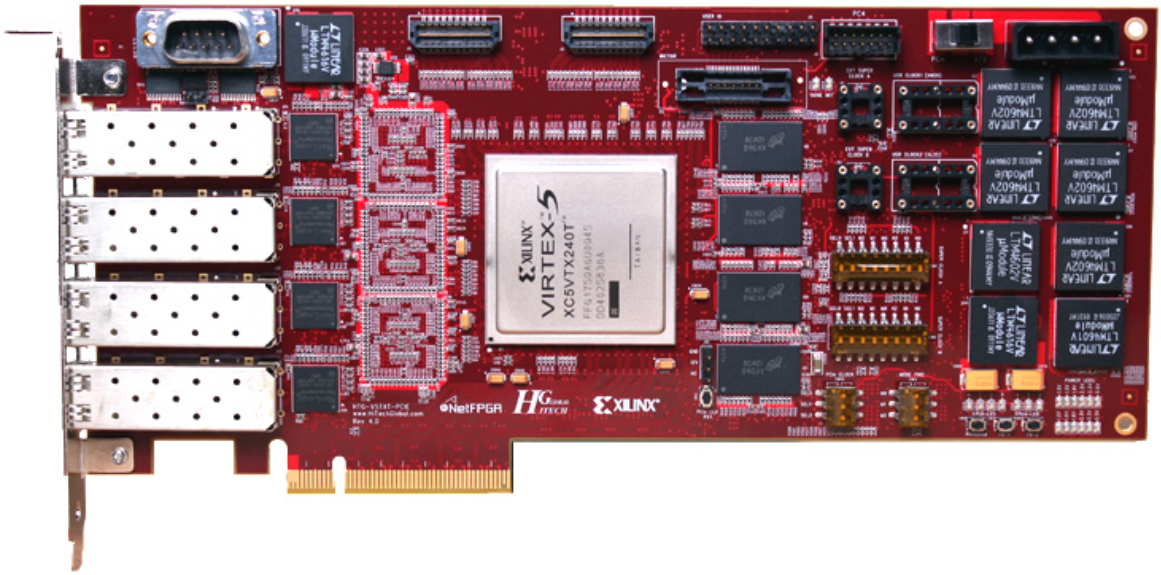


Figure 3.9: NetFPGA 10G development board from the NetFPGA initiative.

3.8 100 Gbit/s Platforms

In this section we present the different platforms we have used during this thesis for 100 Gbit/s network speed. As stated above, Xilinx incorporates a harden 100 Gbit/s Ethernet MAC and Physical Coding Sublayer (PCS) core in its Ultrascale and Ultrascale+ devices, which comply with the IEEE 802.3-2012 specification. Figure 3.10 depicts the CMAC architecture. Note that the interface with the user logic is an LBUS interface. The LBUS interface is 512-bit wide and it is clocked at 322.265625 MHz yielding to a maximum throughput of 165 Gbit/s.

3.8.1 AXI4-Stream adapter

The output communication of the CMAC [61] with the user side is done through a Local BUS (LBUS). LBUS has four 128-bit data segments each of them with independent signalization at 322.265625 MHz. Giving a maximal theoretical throughput of 165 Gbit/s. Xilinx uses this bus because it has less penalty when compared to AXI4-Stream. For instance, in the last transaction the LBUS could misspend 15-Byte out of 16-Byte, however AXI4-Stream could waste 63-Byte out of 64-Byte. However, to facilitate interaction with existing IP-Cores (most of the Xilinx's and third-parties IP-Cores support AXI4-Stream), we developed a LBUS to AXI4-Stream adapter (and vice-versa).

Even though the effective throughput in the AXI4-Stream channel is lower than LBUS in most cases, we can assure than even for the worst case scenario in both ways —

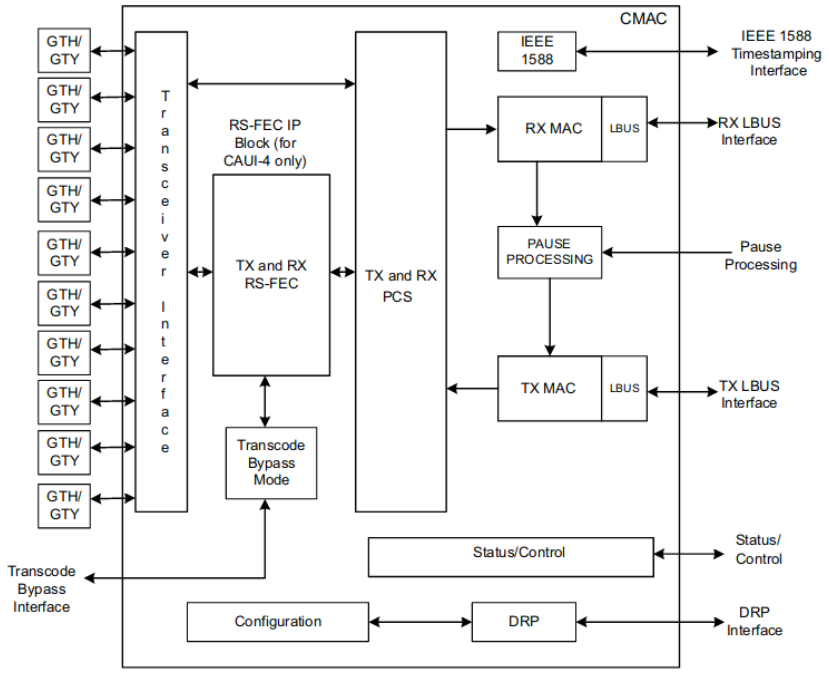


Figure 3.10: Xilinx's integrated CMAC block for 100 Gbit/s. [5]

transmission and reception, the Ethernet effective throughput is met using AXI4-Stream. Note, in an AXI-Stream interface the packets are aligned to each transaction (we use the same clock frequency of 322.265625 MHz). For instance, the worst case scenario is a packet with 65-Byte — Ethernet frame without Frame Check Sequence (FCS) — because we only use 65-Byte out of 128-Byte available, we can compute the throughput as equation 3.1 shows, the constant that multiplies the relation is the maximum AXI4-Stream throughput, which is the same as LBUS mentioned previously.

$$AXI4-S_{Thr} = \frac{65}{128} * 165Gbit/s = 83.789Gbit/s \tag{3.1}$$

On the other hand, the Ethernet effective throughput for a 65-Byte packet is described in equation 3.2, the frames in Ethernet have an overhead of 24-Byte due to preamble (7-Byte), start of frame (1-Byte), FCS (4-Byte) and inter packet gap (12-Byte), those bits are present in the wire but not in the logic.

$$Ethernet_{Thr} = \frac{65}{65+24} * 100Gbit/s = 73.03Gbps \tag{3.2}$$

The previous equation shows that in the worst case scenario there is a margin of at least 10 Gbit/s in the AXI4-Stream interface when compared with the theoretical maximum of Ethernet for a 65-Byte packet. What is more, in Figure 3.11 both LBUS and

[5] Source: https://www.xilinx.com/support/documentation/ip_documentation/cmac_usplus/v2_4/pg203-cmac-usplus.pdf

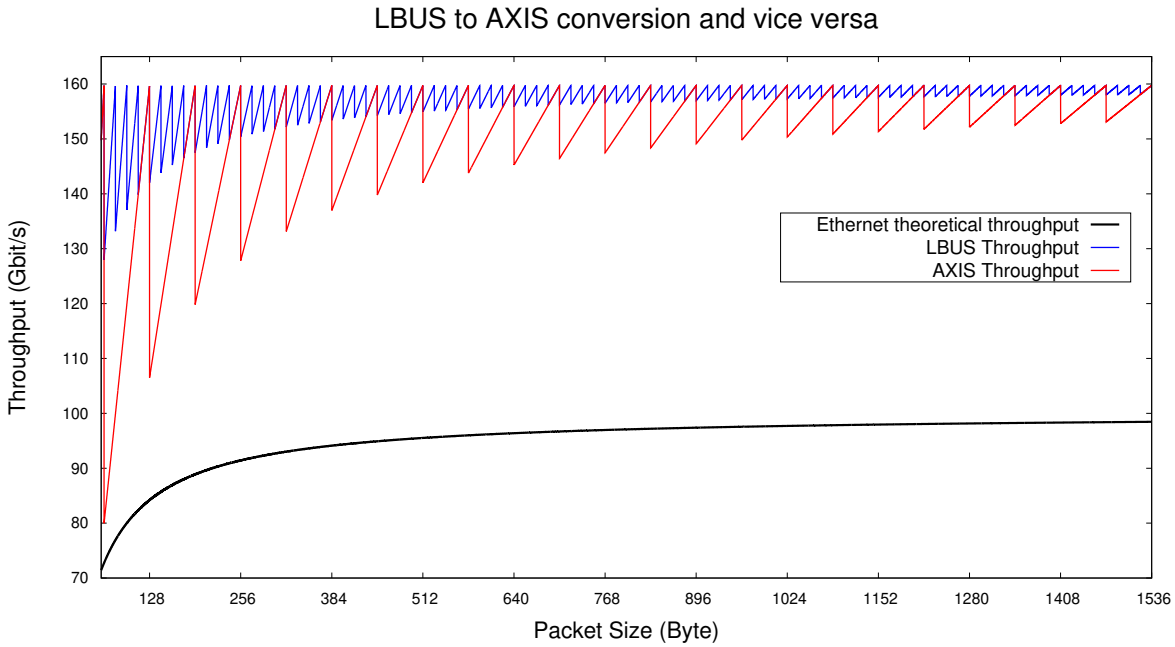


Figure 3.11: LBUS to AXI4-Stream adapter.

AXI4-Stream throughput are summarized as a function of packet size, ranging between 60 and 1536-Byte, which is the typical range in a network. The theoretical baseline of 100 GbE is also shown as a reference. The critical point is a 65-Byte packet, when the AXI4-Stream reaches a performance of 50.78% of the full capacity — we have studied this packet size in detail before. The performance loss for using AXI4-Stream instead of LBUS is a little price to pay when compared with out-of-the-box integration with the rest of IP-Cores provided by Xilinx and third-parties. Consequently, we decided to use AXI4-Stream in our designs. What is more, since Vivado 2019.1 the IP-Core incorporate AXI4-Stream as an interface with the user logic.

VCU108

This board has an Ultrascale device (XCVU095-2FFVA2104E) [62]; one QSFP28 cage and one CFP2 cage both of them support 100 GbE. Additionally, there are two banks of 2.5 GB each of DDR4 memory. Regarding to the programmable logic there are over a million flip-flops, over a half million of LUTs and 1,728 BRAMs. Finally, the board has a PCIe of third generations with eight lanes. Figure 3.12 shows the board.

VCU118

This board has an Ultrascale+ device (XCVU9P-L2FLGA2104E) [63]; two QSFP28 cages both of them support 100 GbE. Additionally, there are two banks of 4 GB each of DDR4 memory. Regarding to the programmable logic there are over two million flip-flops,

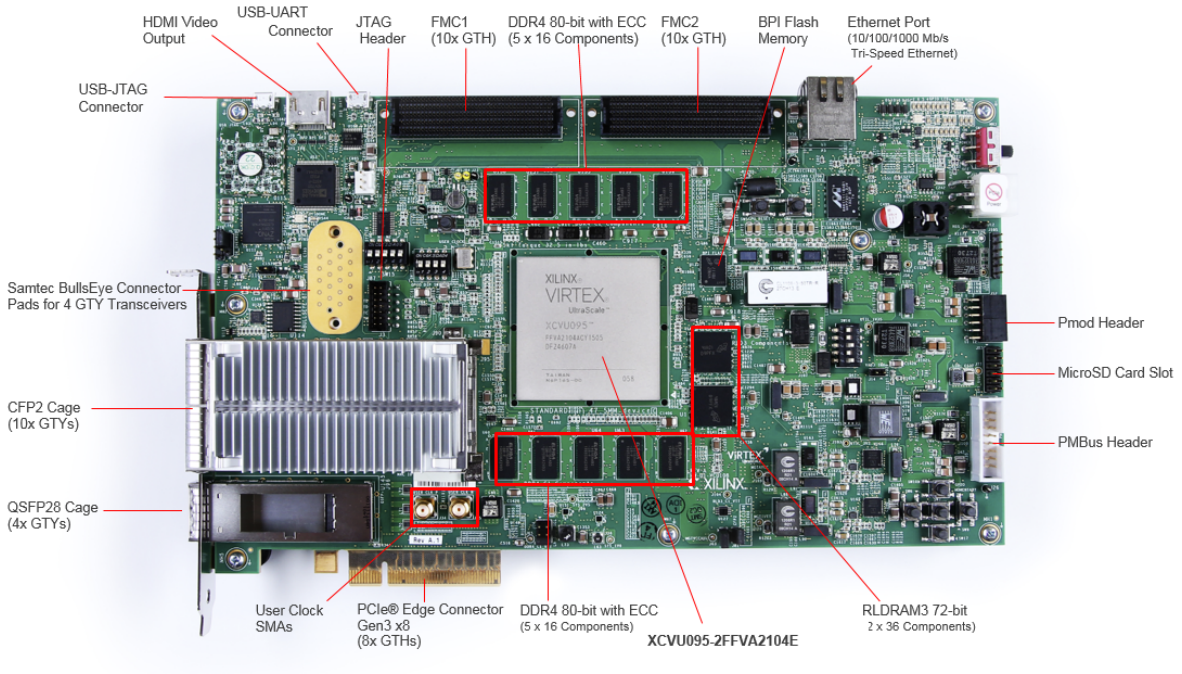


Figure 3.12: VCU108 development board from Xilinx.

over a million of LUTs, 2,160 BRAMs and 270 Mb of URAM memory. Finally, the board has a PCIe of third generations with sixteen lanes. Figure 3.13 shows the board.

ALVEO U200

This board has the same FPGA as the VCU118, however it has 64 GB of memory split into four banks. It also has two QSFP28 cages both of them support 100 GbE. Additionally, it is thought for Data Centers, therefore, it is able to work twenty four hours a day, seven days a week; all the time. Figure 3.14 shows the board.

ADM-PCIE-9V3

The ADM-PCIE-9V3 high-performance network accelerator card includes two QSFP28 cages, 8 GB of DDR4 memory and a XCVU3P-2-FFVC1517 FPGA. The programmable logic is a third part of the VCU118. Figure 3.15 shows the board.

Finally, table 3.2 details the characteristics of the six platforms used during this thesis. Bear in mind that, VCU118 and ALVEO U200 share the same FPGA, therefore there is only one column for both boards, however, the ALVEO U200 has 64 GB of memory.

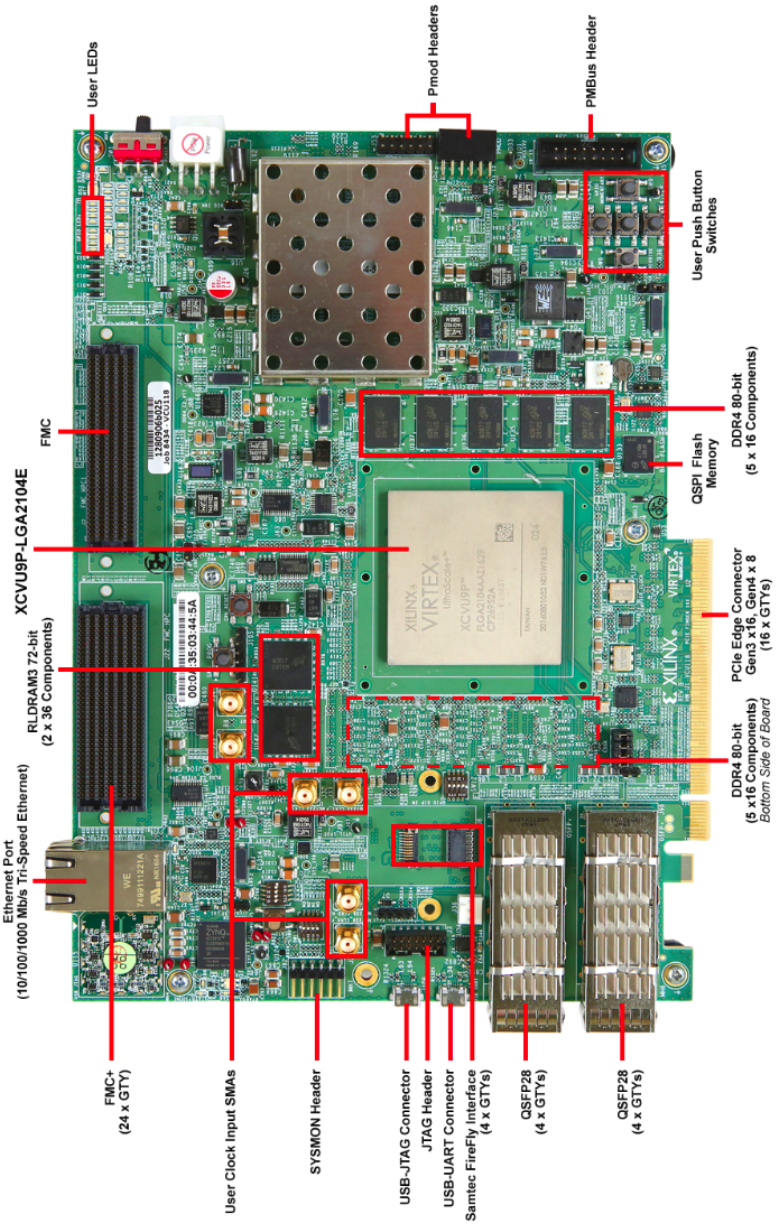


Figure 3.13: VCU118 development board from Xilinx.

Figure 3.14: Alveo U200 data center accelerator card from Xilinx.

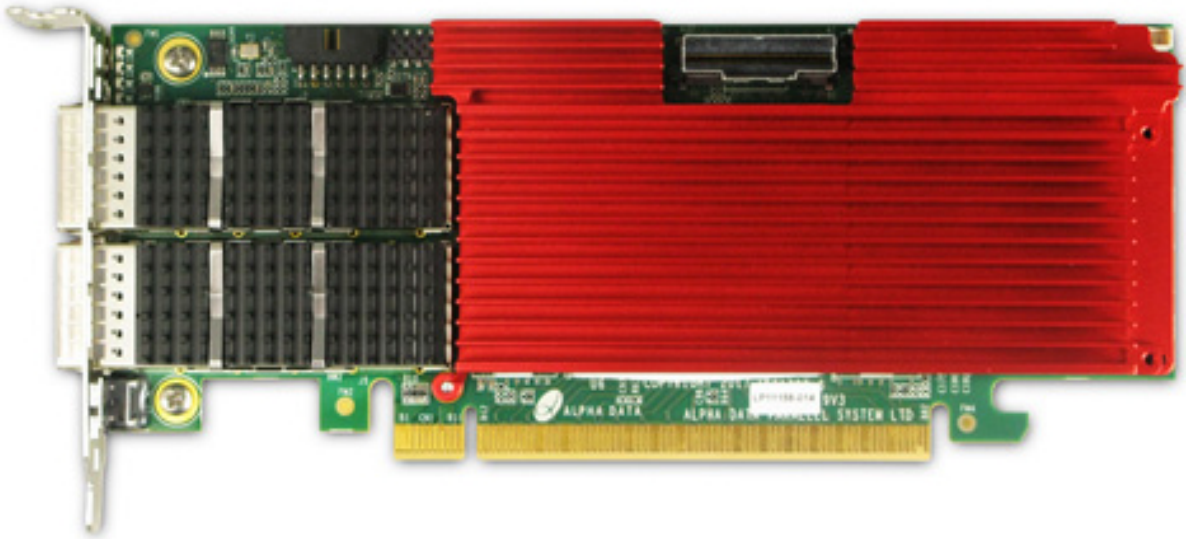


Figure 3.15: ADM-PCIE-9V3 half-length and low profile high-performance network accelerator card from Alpha Data.

Element	ZedBoard	NetFPGA10G	VCU108	VCU118 ALVEO U200	ADM-PCIE-9V3
Device	XC7Z020	XC5VTX240T	XCVU095	XCVU9P	XCVU3P
Node (nm)	28	65	20	16	16
Family	Artix 7	Virtex 5	Kintex Ultrascale	Kintex Ultrascale +	Kintex Ultrascale +
LUT	53,200	74,880	537,600	1,182,000	394,000
FF	106,400	149,760	1,075,200	2,364,000	788,000
BRAM (Mb)	4.9	11.6	60.8	75.9	25.3
URAM (Mb)	N/A	N/A	N/A	270	90
DSP48E	220	96	768	6,840	2,280
Ethernet Speed	1 GbE	10 GbE	100 GbE	100 GbE	100 GbE
MAC	Soft IP	Soft IP	Harden IP	Harden IP	Harden IP
PCI Express	N/A	Gen 2 x8 up to 40 Gbit/s	Four Gen1/2/3	Six Gen3/4 x16/x8	Two Gen3/4 x16/x8
External DRAM	512 MB DDR3	Four x32 RLDRAM II	Two 2.5 GB DDR4	Two 4 GB DDR4 Four 8 GB DDR4	Two 8 GB DDR4
Price (dollars)	$ 449	$ 1,675	$ 5,995	$ 6,995	$ 4,950

Table 3.2: Summary of different features per each platform.

Part II

Network Monitoring

CHAPTER 4

ACTIVE MONITORING

Active monitoring is essential for assessing network infrastructure. Therefore, in this chapter we evaluate FPGAs capabilities to implement such probes. What is more, this chapter shows the advantages of FPGA-based active monitoring probes in terms of accuracy and reliability when comparing the results to those of software-based. Likewise, results show that FPGA-based solutions are significantly better, especially at 10 Gbit/s — we only compare 1 and 10 Gbit/s FPGA implementations against software. Furthermore, this chapter also demonstrates that active monitoring probes using FPGAs are fully scalable — a 1 Gbit/s, a 10 Gbit/s and finally a 100 Gbit/s probes were implemented — while keeping the same level of accuracy. We take advantage of high-level synthesis and open-source platforms, as much as possible, to develop prototypes quicker. Noticeable advantages of our proposal are the high accuracy, the competitive cost with respect to the software counterpart, which runs in high-end off-the-shelf workstations and the capability to easily scale to a wide variety of network speeds.

4.1 Introduction

Active monitoring requires the injection of artificial traffic to the network. One of its advantages is that the control is total in the nature of the injected traffic, which turns out to be extremely useful for a wide variety of scenarios [55], thus, gaining insight of the network behavior. The main drawback of this approach is that extra traffic is introduced into the network which might interfere with real traffic. There are two

important factors that affect real traffic when using active probes: the size and frequency of the artificial probing traffic. Therefore, a trade-off is mandatory to obtain a meaningful estimation without jeopardizing the normal functioning of the network. Moreover, active probes fulfill two purposes: a) measuring Key Performance Indicators (KPI); b) evaluate particular cases in order to foresee issues before they even happen with real traffic. Hence, verifying the Quality of Service (QoS) become rather easy using such type of monitoring probes.

The most basic figures of merit of network links and network equipment are [64]: throughput, latency — One-Way Delay (OWD) or Round-Trip delay Time (RTT) —, jitter and Packet Lost Ratio (PLR). As link speed grows, measuring these parameters is not only more difficult, but also, it calls for testing devices that must provide unprecedented accuracy. For instance, the transmission of a minimum-size Ethernet frame takes 670 ns at 1 Gbit/s; 67 ns at 10 Gbit/s and only 6.7 ns at 100 Gbit/s. Apart from the difficulties of measuring at such small timescales, switching time is no longer negligible, making it imperative to include the switching equipment within the test scope of active probes as well.

Among the different active measurement methods, the packet-train technique has gained interest in the recent years. Such technique not only allows for an accurate measurement of the link throughput, without significantly interfering the existing traffic in the link. But also, it has proven to be effective and highly immune to interferences such as cross-traffic load at the end-user equipment [65]. Section 4.5 further details the packet-train technique, which also provides OWD or RTT and PLR.

Software-based active probes, such as the packet-train technique, work reasonably well for moderate link speeds (up to 1 Gbit/s). The packet-train technique requires a very precise timestamping of incoming and outgoing packets in order to provide an accurate throughput and latency measurement. The faster the link, the more precise the timestamping should be. However, the precision of software-based solutions is known to be limited by uncertainties at the different elements in both the reception and transmission chain (NIC, PCIe bus, operating system). Moreover, current software tools are severely constrained when it comes to performing this type of measurements at high-speed (e.g. 10 Gbit/s), even if they run at kernel level in the operating system. On the other hand, hardware-based solutions have proved to be very accurate, but often criticized when compared to software solutions due to the high skills required to start a new design and the long development cycles (low productivity) as well as acquisition costs. Fortunately, the new FPGA families, as well as High-Level Synthesis (HLS) tools [15, 66] can very efficiently reduce these costs, consequently, making hardware solution a reality in the testing infrastructure.

Currently, the speed of Ethernet interfaces in the state-of-the-art FPGAs goes up to

400 Gbit/s, thus making it possible to use the proposed monitoring approach at both the access and the transport networks. That is, the network quality monitoring devices will be placed at the network ends as well as in the intermediate transport infrastructure. This way, it is possible to have a complete picture of the operation of the network. However, in order for this strategy to be fully usable, all network monitoring devices at access level should be time-synchronized, so that a complete delay map of the network can be constructed. However, devices at transport level could avoid time-synchronization and measure RTT instead.

To evaluate FPGAs capabilities to perform active monitoring, we have followed two different approaches. 1) For 10 Gbit/s, we have used the open-source NetFPGA project. In order to reduce development time, we used NetFPGA-10G [67] OSNT (Open Source Network Tester) [33] monitor project as a starting point, and we also used HLS tools (namely, Vivado-HLS) [68] to implement the IP core in charge of computing network parameters. 2) For 1 and 100 Gbit/s we have used standard boards without networking reference designs. Therefore, we have done the design from scratch using Hardware Description Language (HDL) when determinism was needed, and Vivado-HLS when productivity was sought. Such two design strategies turned out to be very effective both in terms of coding productivity and accuracy of measurements. Actually, the quality of measurements was found to be much better than that obtained with software solutions running on commodity servers, though the development effort was not significantly higher, thanks to the use of open source platforms and HLS tools.

Hence, we propose the use of novel FPGA design as a means for comprehensively testing network equipment with packet trains. The most remarkable novelties are: First of all, we show that very accurate results can be obtained in 1, 10 and 100 Gbit/s, using proof-of-concept designs. Secondly, FPGA SoCs or MPSoC can be used to implement very cost-effective testing appliances, featuring a minimal component footprint and a reduced power consumption, which could easily be deployed across the whole network. Moreover, we also quantify how inaccurate software-based solutions can be at multi-gigabit-per-second speeds, unless the corresponding hardware aid comes into play. Finally, we also show the benefits of open-source platforms and high-level synthesis in order to reduce FPGA development time and cost, thus making programmable logic competitive to software in terms of design productivity.

4.2 Proposed Monitoring Device Architecture

The proposed network quality monitoring device is presented in Figure 4.1. The appliance (FPGA + interfaces + peripherals) will have at least two network interfaces:

one for management, connected to the processor side, and the other for quality monitoring, connected to the FPGA side. The management interface will typically be implemented as 100 Mbit/s or 1 Gbit/s Ethernet. Regarding to the monitoring interface, its speed will match that of the network link to be monitored. Currently, FPGAs support Ethernet speeds up to 400 Gbit/s. In the programmable logic, both the Ethernet MAC and the packet generators and receivers will be implemented. The packet generator will be used for active measurements, whereas the packet receiver is needed in case of passive measurements (chapter 5). A key element in the monitoring devices is the time clock synchronization block, which enables one-way delay measurements. It will take the input of a GPS receiver as reference to generate a local clock synchronized to Universal Time Coordinated (UTC). What is more, quality monitoring devices will be located at the network edges, at the client connections. Monitoring devices will also be located at intermediate routing/switching locations of the network. The devices located at the edges will perform active measurements, whereas the devices at the intermediate points will mainly be dedicated to passive measurements. All devices will be controlled by a centralized monitoring entity, which will be in charge of generating a map of the situation of the network, identifying possible malfunctions and/or bottlenecks. The advantage of having monitoring devices at various points of the network is that this allows for the construction of a complete quality map of the network. However, the challenge here is the economic costs of such comprehensive deployment of monitoring devices, which is in part tackled by this work.

The processor and the blocks implemented in the FPGA side (Ethernet MAC, packet

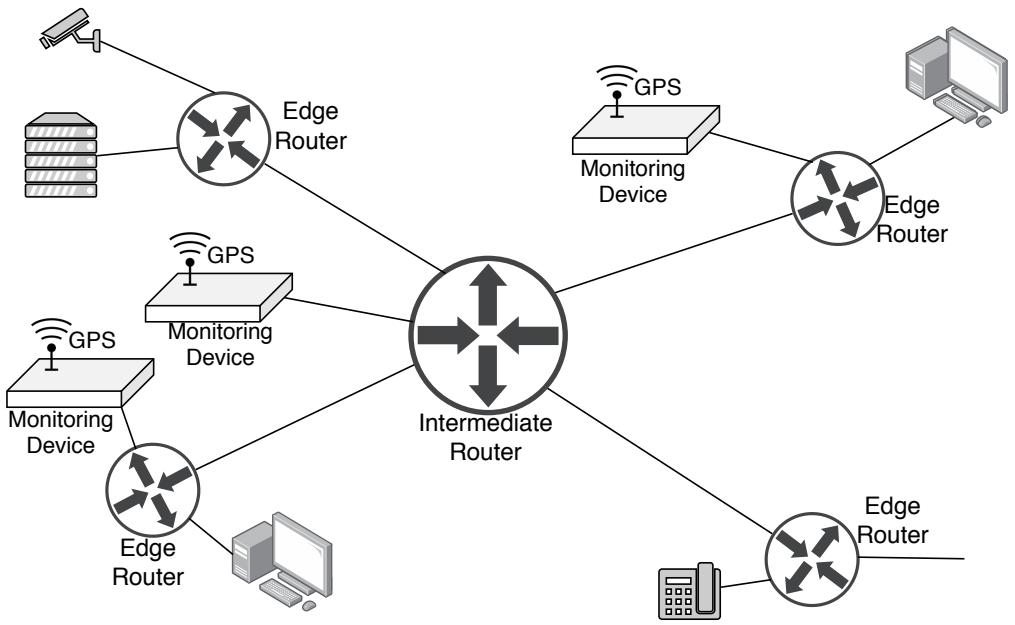


Figure 4.1: Device monitoring in different parts of a network.

generator and receiver, clock synchronization) will be connected via a standard AXI4 interface. Apart from the programmable SoC, the other components needed in the network monitoring device are Double Data Rate (DDR) memory, Flash memory, GPS receiver, Ethernet physical medium devices and power regulators. That is, the monitoring appliances will feature a minimal component count, thus allowing for low-cost solutions to be developed. Moreover, the proposed solution features a small size and reduced power consumption, for instance, 8.5 W in our proof-of-concept design for 1 Gbit/s interfaces.

4.3 Related Work

Certainly, the benefits of using FPGAs in multi-gigabit-per-second networks are well-known. However, previous works in the active network testing area are scarce and mainly focus on replacing configurable traffic generators [69–71]. Probably the closest proposal to ours is [72]. Such work uses FPGAs to test networking equipment according to RFC2544 [73], which aims at benchmarking methodology for network interconnect devices. However, such work considers only 1 Gbit/s networks, and no previous work considers accurate time synchronization mechanisms such as GPS, which is needed to accurately measure OWD and jitter in a distributed way. Finally, the benefits of using HLS for networking applications are beginning to be recognized, as shown in [30] where such methodology has been used to implement a 10 Gbit/s TCP/IP stack in FPGA.

Active monitoring is a well-known tool to evaluate network performance. In this context, there are diverse open-source alternatives in GNU/Linux, such as `pktgen` [74], which implement a packet generation at kernel-level with the possibility of timestamping these packets. Such software allows implementing the packet-train network measurement technique, which is later explained in subsection 4.5. Intel DPDK [75] is a user-level framework which allows operating in network links up to 100 Gbit/s, but at this moment DPDK does not include packet timestamping. Another problem with software solutions is that it is becoming more challenging for conventional server architectures to monitor multi-gigabit-per-second networks. Not only that, such high-end servers are very expensive and some times the price is comparable with an FPGA card able to work at the same link speed.

With the purpose of implementing cost-effective monitoring devices, it is mandatory to move to the FPGA arena, leveraging programmable logic for deterministic timing and software for remote access and management, running on the embedded processor or host machine. Since some years ago, there are open-source platforms, such as NetFPGA, which provide a network-oriented framework for rapid prototyping of designs and they have become a reference in research. In this scope, several contributions based on

NetFPGA-10G (section 3.7) can be found. For instance, the work in [33] presents a framework for passive monitoring. Active measurements have also been implemented in [66], and in this other work [72] the use of NetFPGA to test networking equipment at 1 Gbit/s according to RFC2544 [73] is described as well. However, such works do not study the effects of clock drift in the measurements and attaches the FPGA to a host, thus augmenting the cost and power so that a massive deployment of monitoring devices will not be cost-effective.

4.4 Use Cases

The range of application of high-speed network testing tools is very diverse. The typical use case focuses on testing the capabilities of network equipment. Nonetheless, this testing can also be extended to other scenarios, where it is necessary to perform distributed measurements along a network path. Moreover, such testing must be performed continuously in time to monitor the network Quality of Service (QoS) parameters. Below we provided three examples, in two of them this type of distributed continuous testing has been applied, apart from the usual testing of network equipment. In the third example the measurements must be performed before the path is put into operation.

4.4.1 Service Level Verification

Nowadays, we are witnessing a fierce competition between operators to provide more bandwidth in the residential access link for the less possible price, some ISP offer 10 Gbit/s connection for residential access. In this competition for market share the regulators are playing their role as referees to enforce a given QoS level.

Of particular interest is the case of bandwidth reselling between operators, at the access and metro link level. There, the regulator must ensure that the QoS provided by the incumbent operator to the hiring operator meets a QoS level that allows the transmission of interactive multimedia services. Since the number of potential users in the metro network is very large and the residential bandwidth is growing at a significant pace, see Figure 1.2. We note that the metro network switches, working at multi-gigabit-per-second data rates, must be carefully tested both before deployment and also during operation.

4.4.2 Measurement of Next-Generation Elastic Optical Network Equipment

Elastic optical networks are being developed to offer the possibility of dynamically changing the signal modulation format and/or the spectrum allocation of optical data links [76, 77]. This dynamic reconfiguration capability paves the way for new operation models, whereby links are no longer statically provisioned, but dynamically adapted to traffic demands. Therefore, the link capacity must be continuously monitored in these elastic optical networks, to check if the underlying network has reconfigured the provided bandwidth or not.

4.4.3 Measurement of Dynamically Provisioned Optical Paths

The H2020 Metro-Haul research project [78] aims at defining the metro optical network architecture to interconnect incoming 5G (fifth generation of mobile network technology) access networks with the core network. For this, it is developing new techniques, based on Software Defined Network (SDN), to provision optical paths dynamically in an optical transport network. This metro network has to provide Key Performance Indicators (KPI) such as capacity, latency or packet loss, with very stringent requirements.

In order to verify that the deployed optical paths meet these requirements, it is mandatory to test them before they are put in operation. In such context, active measurements must be performed at 100 Gbit/s to check that the requested KPIs are met before the optical path is put in operation. Given the foreseen heterogeneity in the optical network, with different equipment providers, these preliminary measurements are essential for the network operator to assure the QoS provided to the users, and complement passive measurements performed during the optical path lifetime [79]. Currently, COTS equipment is not able to provide such measurement with the necessary accuracy. Therefore, specialized hardware is mandatory. In this light, FPGAs arise as the most cost-effective solution.

4.5 Packet-Pair and Packet-Train Techniques for Network Measurements

We propose to use the packet-train technique, which is an evolution from the previous packet-pair technique. Packet-pair [80] is an active measurement method based on sending multiple packet pairs from a source to a destination endpoint in order to estimate the corresponding QoS parameters. Each pair is composed of equally sized packets sent back-to-back at the maximum allowed speed in a link or end-to-end path. At the receiver

side, packet dispersion is analyzed to estimate the capacity. As it turns out, packet-pair techniques are prone to both capacity underestimations and overestimations due to interfering traffic, because only two packets are used in the measurements. However, packet-trains [81, 82] provide better accuracy and robustness, simply because more packets are involved in the measurement and the resulting train is less sensible to cross-traffic interference than the corresponding pair.

When using packet-train techniques, a group of N packets is sent back-to-back from a sender to a receiver and the average dispersion of the N packets is used to calculate the capacity, as shown in Figure 4.2. Additionally, One-Way Delay (OWD), jitter and Packet Lost Ratio (PLR) may also be estimated by including timestamps and sequence numbers on the packets. Increasing the number of packets in the train provides immunity against interfering traffic but also increases interference in the measured network. What is more, this technique is based on flooding the link, and, consequently, the measurement time interferes the rest of traffic on the path, so must be kept at a minimum — unless the path is not in service yet, for instance, the scenario described in section 4.4.3. Typically, train lengths range from 100 to 1,000 packets. Regarding packet sizes, OWD is better measured using minimum-sized packets to reduce the impact of the transmission time on the estimation. Yet, for OWD probes must be time-synchronized, thus increasing the complexity in the implementation. However, Round-Trip delay Time (RTT) can be measured using a packet reflector at the end of the path to measure, hence, avoiding time synchronization.

On the other hand, the confidence level in the measurement plays a key role in the number of packets within the train. But as shown later, for hardware implementations short number of packets are as confident as larger number of them. Nevertheless, depending on the current state of the path or device under test, the number of packets within the train must be traded-off to reduce interference with real traffic.

Equation (4.1) shows how latency is obtained, where TSr_i is the timestamp of the packet i at the receiver, and TSt_i is the timestamp of the packet i at the transmitter. Equation (4.2) presents the expression to calculate instantaneous link throughput, where L is the frame size. Finally, equation (4.3) shows how to calculate jitter. To correctly perform the calculations in (4.2) and (4.3), it must be verified that packets are consecutive, as this is implicit in these equations.

$$latency = TSr_i - TSt_i \tag{4.1}$$

$$throughput = \frac{L}{TSr_i - TSr_{i-1}} \tag{4.2}$$

$$jitter = ||TSr_{i-2} - TSr_{i-1}| - |TSr_{i-1} - TSr_i|| \tag{4.3}$$

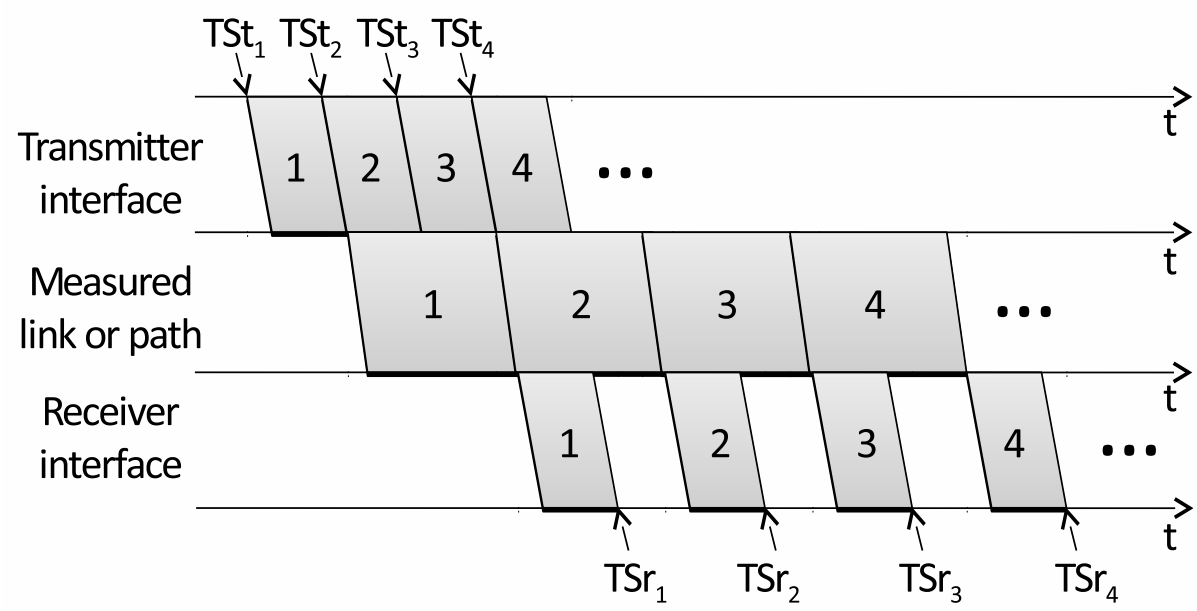


Figure 4.2: Representation of packet-train technique.

4.6 Software-based Solutions

Traditionally, network measurements have been performed using specialized hardware designed for such a task. In the recent years, several software-based solutions that run on top of Commercial Off-The-Shelf (COTS) systems have also been applied for network measurement and testing tasks. The latter provide a cost-effective and flexible solution for the development of network testing probes. For instance, `pktgen` [74] is a Linux kernel module that enables the generation of traffic with different packet headers and payloads defined by the user (source and destination MAC and IP addresses, UDP ports, etc.) and also with specific statistical features, namely inter-arrival time and number of flows. Additionally, this kernel module adds a sequence number and the departure timestamp for each packet, which makes this tool suitable for throughput, OWD, jitter and PLR measurements. The main drawback is that the departure timestamp is taken in the Linux kernel and not in the Network Interface Card (NIC) itself, which adds measurement noise due to the transit time from the Linux kernel to the NIC. In high-speed links (10 Gbit/s and beyond), it is noticeable that more packets per second must be copied to the kernel, so the measurement noise is more significant. Thus, all traditional software traffic generators cannot exactly mimic the transmission pattern defined by the user, which severely biases the measurement in a high-speed scenario, as stated in [83]. Even if a real-time operating system is used, the interruption timer accuracy is in the order of milliseconds, that is far too coarse for 10 Gbit/s networks and even worse for 100 Gbit/s.

In addition, at the time of this work was done, vanilla network drivers could not cope with minimum-sized packets at 10 Gbit/s rates neither in transmission nor in reception, which is essential for testing. Recently, high-speed network engines have been developed [84] to solve this issue. For instance, a software traffic generator called *PktGen-DPDK* [85] built on top of Intel's DPDK [75] is available. Such traffic generator is able to transmit either random generated packets or PCAP traces. Although this traffic generator provides 10 Gbit/s and beyond rates, at the time of the work was done it could not add sequence numbers neither transmission timestamps to the packets, so it was not able to measure OWD. Recently, DPDK-LatencyMetter has been released [86] for 10 Gbit/s network, such work achieves really good result at the expense of sending thousand times each train, thus flooding the network for a long period of time. Only the Linux `pktgen` module will be considered later in our comparative analysis for 1 Gbit/s and 10 Gbit/s.

4.7 Hardware-based Solutions

For years, the use of programmable logic devices (more specifically, FPGAs) has democratized hardware design for low volume users [53]. Nevertheless, the complexity of designing specialized hardware still resides in the FPGA design flow, which traditionally has been based on the challenging Hardware Description Language (HDL). To circumvent this issue, Vivado-HLS tool was used section 3.3.2. HLS typically uses C/C++ as design entry, instead of the lower-level Register Transfer Level (RTL) abstraction model used by HDLs. HLS tools not only improve design productivity, but also bring FPGA technology closer to networking engineers.

We take advantage of Vivado-HLS to develop several hardware-based solutions. Three designs have been implemented, one for each network link speed: 1 Gbit/s, 10 Gbit/s and 100 Gbit/s. The first one was used to demonstrate the feasibility of building accurate, affordable, low power and portable 1 Gbit/s network testing appliances on top of an FPGA SoC device. The second one, provided as proof-of-concept for 10 Gbit/s networks and it is based on the open-source NetFPGA [87] project. Finally, the third one is built leveraging the Virtex Ultrascale+ FPGA-fabric and provides the proof-of-concept for 100 Gbit/s links, which is the current trend for backbone and metro links. For the sake of simplicity, they are going to be referred as HwP1, HwP10 and HwP100 respectively.

Time synchronization is not a straightforward task. In local area networks, protocols such as Precision Time Protocol (PTP) can be used. But PTP is no longer valid for Internet-wide measurements (it requires that all intermediate routers/switches support PTP), so other approaches should be used. The most common alternative is to use Global

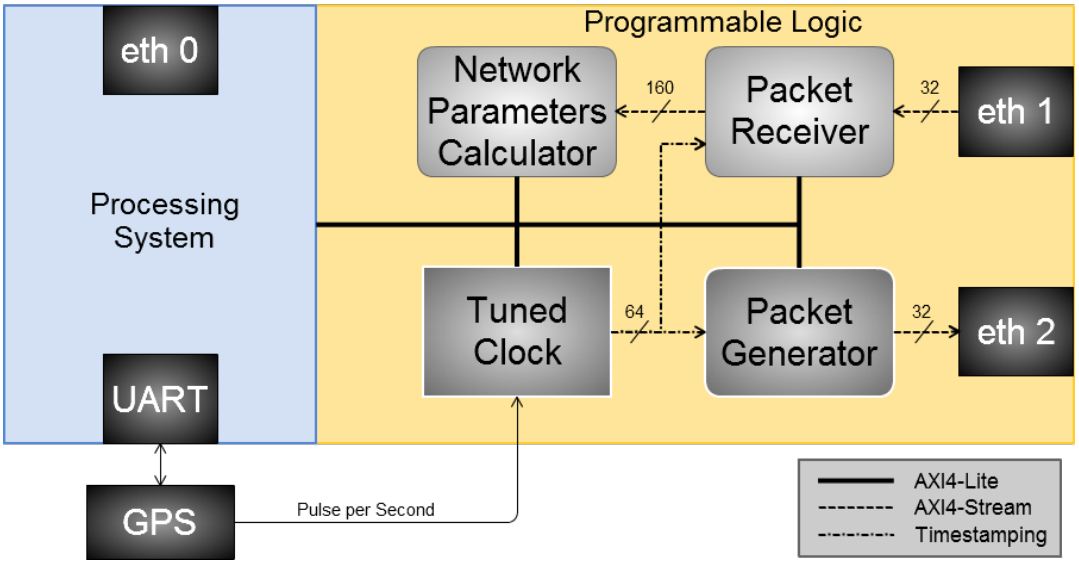


Figure 4.3: High level overview of the ZedBoard (SoC) design.

Positioning System (GPS) receivers, since GPS provides a precision close to that of an atomic clock at a reasonable price. However, it remains a challenge to utilize a GPS clock reference in a network monitoring device while maintaining the low-cost and low-power features.

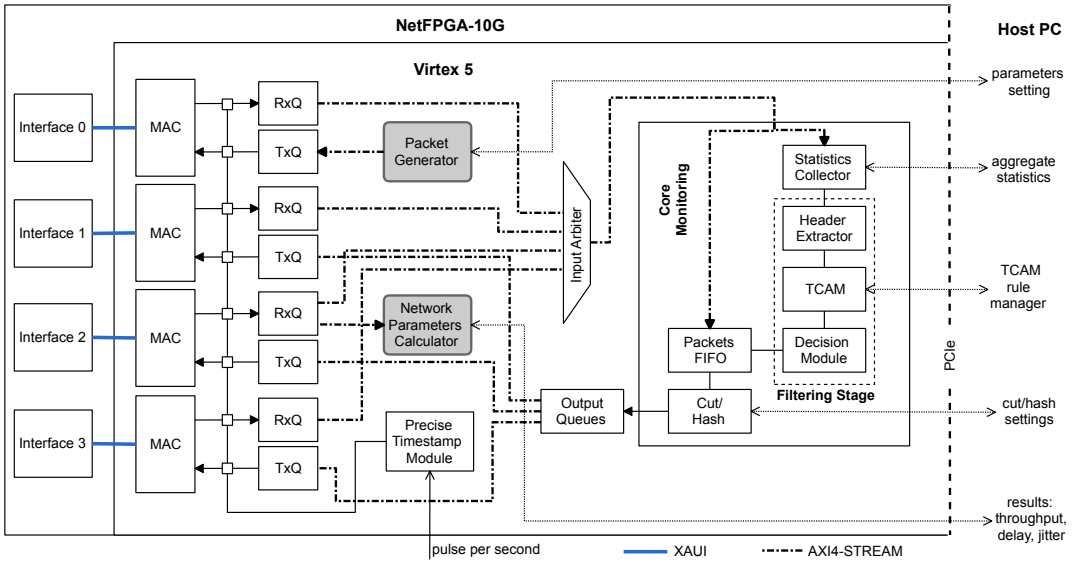


Figure 4.4: High level overview of the NetFPGA-10G design, based on the open-source network tester project.

4.7.1 Hardware Architectures Description for HwP1 and HwP10

With the aim of speeding up the hardware development cycle and bringing it as close as possible to the network application engineers, we have used HLS tools to design the prototypes, as described in [15]. Our tool of choice is Vivado-HLS [68], which generates synthesizable HDL code from a C/C++ source along with synthesis directives. On the other hand, modules where timing is critical (operations need to be done in an exact number of clock cycles) were implemented using the traditional FPGA design flow based on HDLs (VHDL or Verilog).

The designs communicate with the external Physical Layer (PHY) chip by means of an AXI4-Stream interface. The difference between both prototypes is in the width of the bus and its frequency of operation (32-bit at 100 MHz for HwP1 and 256-bit at 156.25 MHz for HwP10).

In both architectures, depicted in Figures 4.3 and 4.4, there are two key IP-Cores with similar behavior in the heart of the system. On the one hand, the **Packet Generator** was developed using HLS for HwP1 and HDL for HwP10, apart from the language, the differences are: 1) AXI4-Stream width and 2) frequency of operation. Customizable options include source and destination MAC and IP addresses, UDP ports, packet size and train length. In addition, each generated packet contains a sequence number, and it is timestamped as close as possible to the physical layer chip. On the other hand, the **Accurate Timestamp Module** is in charge of clock synchronization and clock drift correction. Both implementations use the Pulse Per Second (PPS) signal from a GPS receiver as a reference to compensate the clock drift. At HwP1, this module is split in two parts following a hybrid hardware-software approach: the first part is a variable-rate counter implemented in hardware, which runs at 100 MHz. The second part is an algorithm that runs in the ARM processor. Such algorithm uses the sum of the previous errors to correct the rate of the counter, and sends it back to hardware. Therefore, we can obtain a remarkable timestamp resolution of 10 ns, with an extremely low clock drift, thanks to the GPS-based error compensation. At HwP10, we have leveraged on the OSNT project functionality, which implements a Direct Digital Synthesizer (DDS) to correct the clock drift [33]. The design operates at 156.25 MHz, with a 6.4 ns resolution.

There are some differences in the architectures mainly because HwP1 has a tightly coupled processor near the FPGA (enabling a hybrid hardware-software approach, as mentioned before) and HwP10 provides a framework with a library of pre-designed modules. In HwP1, following the hybrid hardware-software approach, the **Packet Receiver**, developed in HLS, receives the packets and filters them according to user-defined rules. Then, a software program running on the ARM processor computes the **Network**

Parameters.

On the other hand, HwP10 uses the infrastructure available in the NetFPGA framework for packet reception and a custom-developed HLS module to compute the **Network Parameters** which is able to calculate active parameters of a network using a specific data stream based on the packet-train technique. Equations 4.1, 4.2 and 4.3 are implemented in this module. To do so, it receives a 160-bit AXI4-Stream with the needed meta data, which is generated in the reception logic, one transaction for each packet that meets the filtering rules (that is, the packet corresponds to a packet-train measurement). The result can be queried from its AXI4-Lite interface, using an application in the host that reads the network parameters calculator registers.

In summary, the designs are good to send packet trains with the aim of measuring the required quality parameters (delay, jitter, loss and throughput). The logic of HwP1 was fully implemented using HLS, whereas, in the HwP10 only the packet generator was implemented on HDL. Additionally, both designs use a PPS signal from an external GPS in order to support OWD and jitter measurements when testing in a distributed infrastructure. The code of both projects is freely available on GitHub [88].

Finally, it is worth remarking that both hardware implementations occupy less than 55 % of most of the available resources in the FPGA. Absolute figures of used resources in both solutions are shown in Table 4.2. More details about the implementation can be found on the published papers [66, 89, 90].

4.8 Performance Evaluation

4.8.1 Evaluation Testbeds

Two different performance evaluation scenarios have been considered for both software and hardware solutions. The first scenario, used for calibration, is based on sending measurement packets through an interface and receiving them in another interface of the same testing device, in a loopback fashion. The second scenario is based on sending the measurement packets through an interface that is connected to the DUT —in this particular case, a Cisco Catalyst 2960-S. In the DUT, the measurement traffic is forwarded from one SFP+ port to another SFP+ port that is connected to the traffic receiver interface of the testing device. This scenario addresses the 10 Gbit/s case. For the sake of completeness, the performance analysis has been repeated for the 1 Gbit/s case, using the same setup but connected to 1 Gbit/s DUT ports.

To evaluate the software solution, the `pktgen` module has been executed on a server running an Ubuntu Linux 14.04 with a 3.16.0 kernel. The server has two Intel Xeon

E5-2620 processors with 6 cores each, 32 GB of RAM and an Intel 82599 10 Gbit/s NIC. For all tests the `ixgbe` vanilla driver has been used along with `pktgen` 2.75 module.

All the experiments featured packet trains of 100 and 1,000 packets with frame sizes of 60, 64, 128, 256, 512, 1,024 and 1,514 bytes, excluding frame preamble and check sequence. The experiment was repeated ten times to obtain mean and standard deviation for throughput and OWD. In the case of `pktgen` module, traffic has been generated using a single transmission queue since packet sequence numbers must be correlative for throughput and delay measurements, and using multiple queues produces packet disorder [91].

4.8.2 Experimental Results

The throughput measurements in the 1 Gbit/s scenario using the testing setup with the DUT are shown in Figure 4.5. As it can be observed, results obtained with the hardware HwP1 prototype are fairly close to theoretical throughput value and the standard deviation is very small. In the case of measurements using the software approach, if the train length is 100 packets and the packet sizes are lower than 256-Byte, the results are quite far from the theoretical values, and present larger deviations. With sizes of 256-Byte and above, we observe that the empirical values approach the theoretical ones. For 1,000-packet trains, measurements are more accurate and present less deviation.

Likewise, Figure 4.6 shows the throughput measurements for 10 Gbit/s. Similar to 1 Gbit/s, the results for the hardware system are very similar to the theoretical values, with extremely low deviation. However, the results for the software system significantly depart from the theoretical ones, but improve as the packet size and train length increase. From this results, we can conclude that software-based systems are not suitable for 10 Gbit/s active measurement. Similar results were obtained in the calibration setup, measuring the loopback for 1 and 10 Gbit/s scenarios.

In both figures (4.5 and 4.6) the theoretical curve indicates the maximum number useful bits/s that can be carried in a given network and is calculated as $theoretical = \frac{framesize}{framesize+24} * LinkSpeed(\frac{bits}{s})$, the number 24 is the sum of preamble, CRC and inter-frame gap.

Note that, for some scenarios, regulatory bodies require that link measurements are performed using minimum-sized packets, which implies that software solutions are infeasible. Additionally, a thorough network device testing should generate traffic with packet sizes ranging from the minimum size to the MTU.

Regarding OWD measurements, Table 4.1 shows the results for both loopback and switch setups at 1 Gbit/s and 10 Gbit/s. As it can be observed, software measurements are

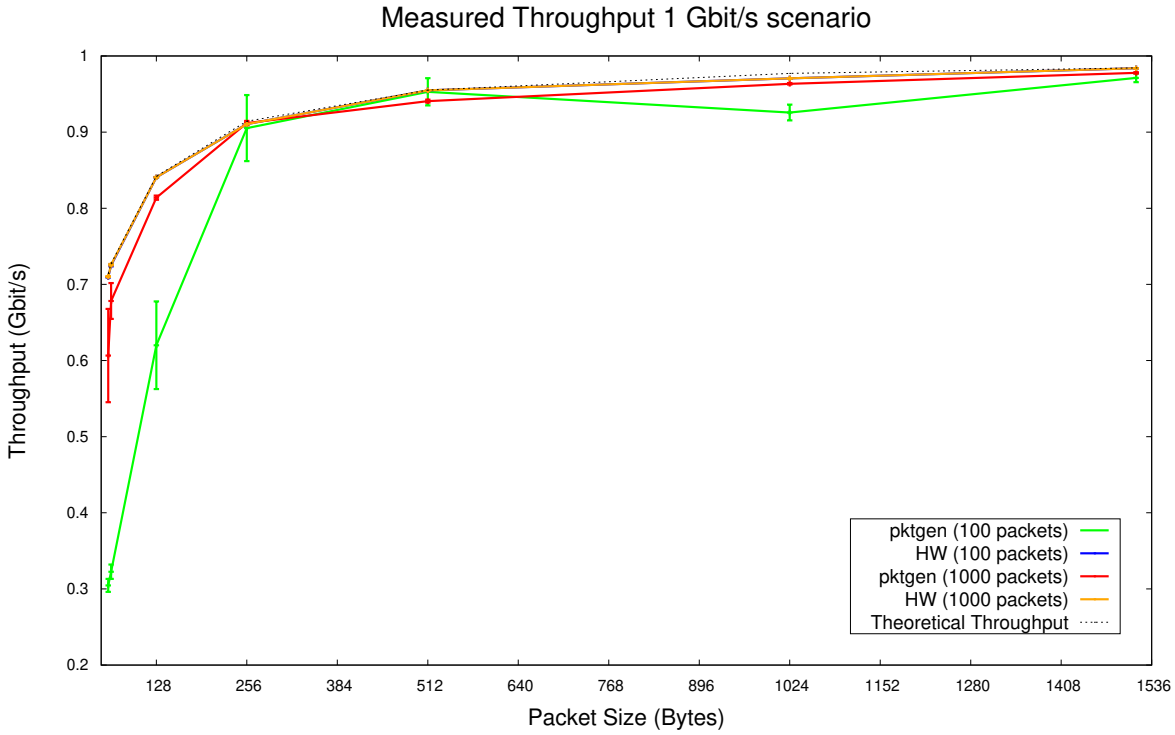


Figure 4.5: 1 Gbit/s DUT throughput with different packet sizes. Mean and standard deviation measured both in software and hardware.

far from the hardware values, adding up to 150 μs of error in the worst case scenario — loopback at 1 Gbit/s. If accuracy below thousands of μs is needed, the hardware solution is the most suitable option. It is also worth noting that the hardware approach not only shows the most accurate results, but it also presents extremely low variation (less than 0.01%) which makes it well suited for jitter measurements. It is also worth to highlight that the confidence level in the measurement of both hardware implementations is extremely high, both train length of 100 and 1,000 packets shows the same results.

4.8.3 Latency Calibration for HwP1 and HwP10

We note that the measured latency in the hardware developments is significantly larger than the theoretical minimum (frame transmission time) in the loopback scenario. This is due to the different elements in the transmission chain. In the 10 Gbit/s case, the FPGA features three IP-cores that add up latency to transmission and reception: 10G MAC core, 10G Attachment Unit Interface (XAUI) core and Multi Gigabit Transceiver (MGT). Particularly, the reception MGT has an elastic buffer in order use the same clock for transmission and reception, so that the design is simplified. Such elastic buffer will also add uncertainty to the latency measurement. Moreover, we note that this buffer is not the only source of latency; the 10G MAC and XAUI cores can add up

to 200 ns (adding transmission and reception latencies). Nevertheless, the main source of latency in the NetFPGA-10G board is the physical medium chip, which performs a conversion from XAUI to the 10 Gbit/s serial electrical interface, as well as Electronic Dispersion Compensation (EDC). Such operations add a significant latency to the transmission/reception path. Similar considerations should be taken into account for 1 Gbit/s case.

Considering the results of Table 4.1, we can empirically infer that the aggregate delays (due to the reasons discussed above) linearly depend on the packet size. Therefore, we can use the loopback scenario to extract a calibration function from the delay measurements. To obtain such a function, we have firstly represented the scatter plot of measured data set (70 points) as shown in Figure 4.7 for 1 Gbit/s and Figure 4.8 for 10 Gbit/s. As a second step, we have fitted the data using a linear regression, as it is the simplest method to calculate such function. Finally, we have subtracted this function from the theoretical one to obtain the calibration function. Such calibration function has been later applied to the Catalyst 2960-S switch delay measurements, obtaining comparable results to those reported [92] for a similar device.

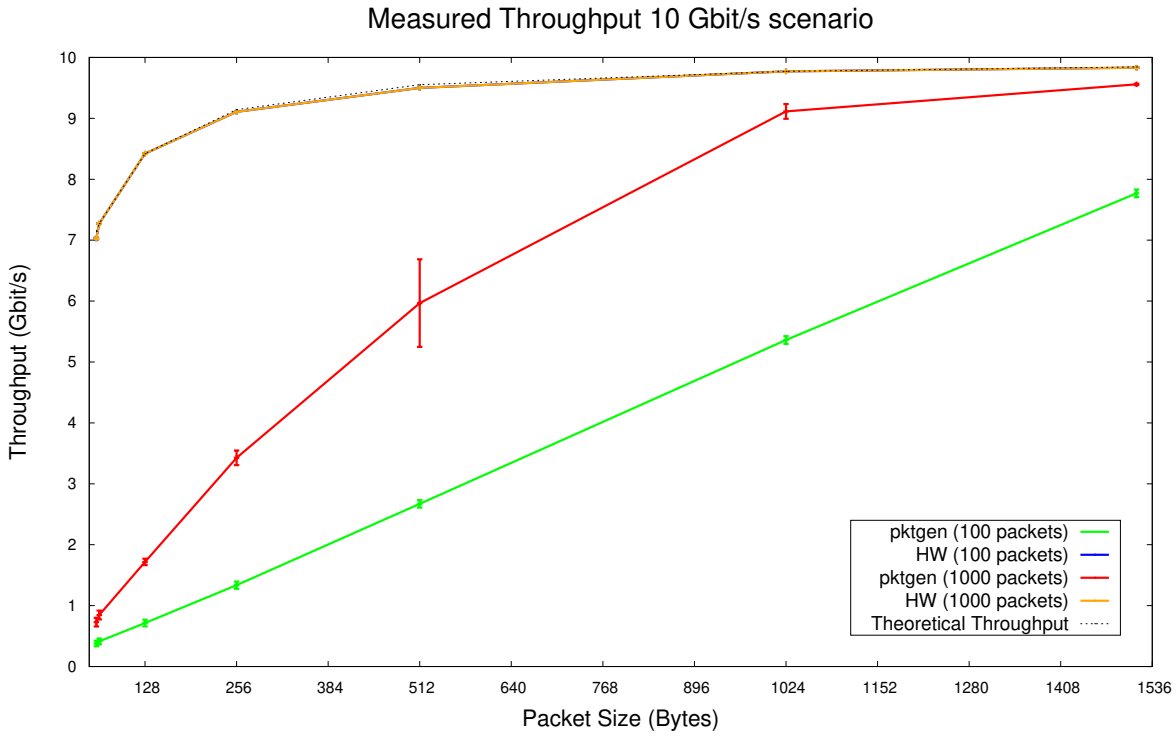


Figure 4.6: 10 Gbit/s DUT throughput with different packet sizes. Mean and standard deviation measured both in software and hardware.

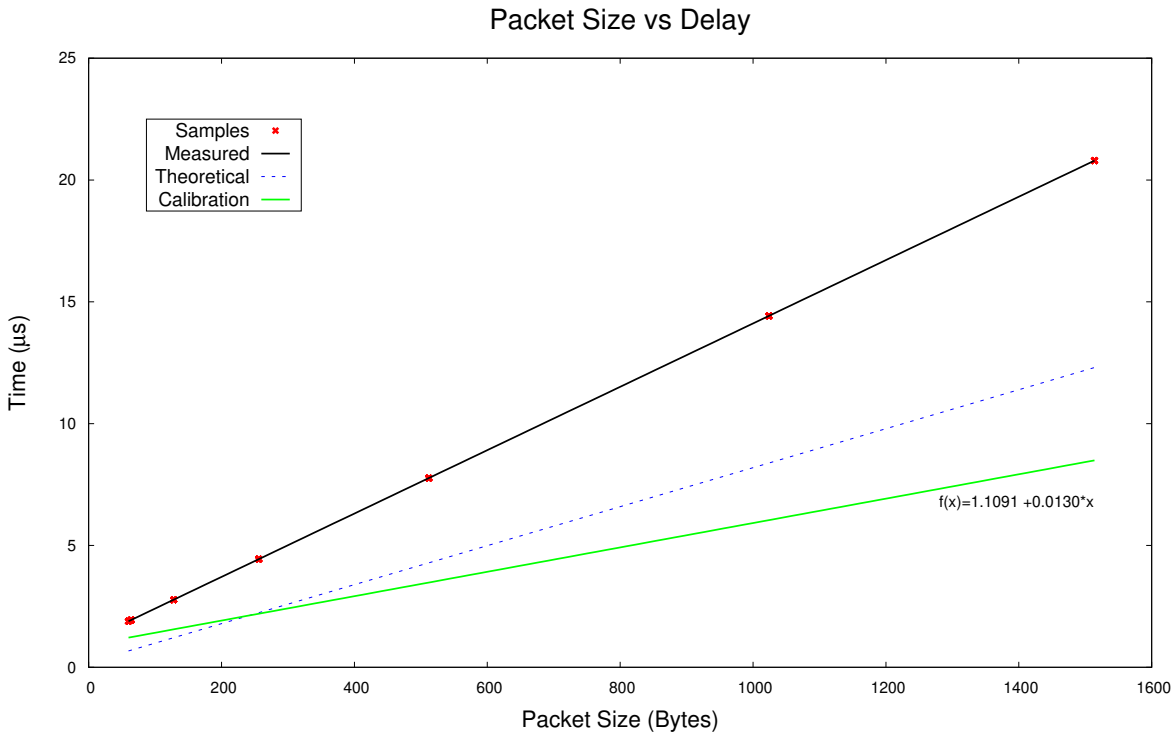


Figure 4.7: Regression on hardware platforms to calibrate measured delay on ZedBoard at 1 Gbits/s.

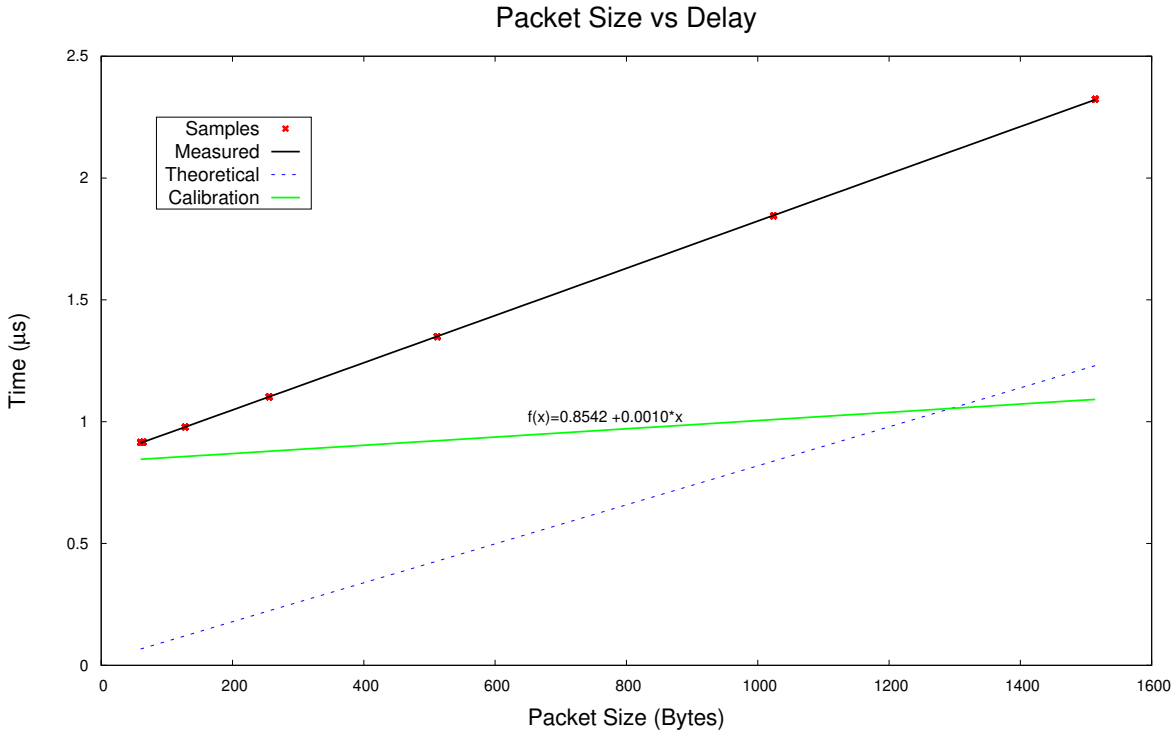


Figure 4.8: Regression on hardware platforms to calibrate measured delay on NetFPGA and OSNT at 10 Gbit/s.

		1Gbit/s				10Gbit/s			
		Loopback		Switch		Loopback		Switch	
Packet Size (bytes)	**# of packets**	**OWD *pktgen* (μs)**	**OWD HwP1 (μs)**	**OWD *pktgen* (μs)**	**OWD HwP1 (μs)**	**OWD *pktgen* (μs)**	**OWD HwP10 (μs)**	**OWD *pktgen* (μs)**	**OWD HwP10 (μs)**
60	100	29 ± 5	1.890 ± 0.003	38 ± 14	5.149 ± 0.003	17 ± 6	0.883 ± 0.000	17 ± 8	3.79 ± 0.003
	1,000	30 ± 7	1.889 ± 0.000	34 ± 5	5.149 ± 0.001	17 ± 5	0.883 ± 0.000	17 ± 6	3.80 ± 0.002
64	100	33 ± 9	1.941 ± 0.002	34 ± 6	5.236 ± 0.003	17 ± 7	0.883 ± 0.000	18 ± 5	3.81 ± 0.001
	1,000	30 ± 5	1.945 ± 0.003	33 ± 4	5.235 ± 0.002	18 ± 6	0.884 ± 0.000	18 ± 6	3.80 ± 0.002
128	100	34 ± 5	2.772 ± 0.001	43 ± 6	6.720 ± 0.005	20 ± 5	0.945 ± 0.000	20 ± 3	3.97 ± 0.007
	1,000	37 ± 5	2.773 ± 0.003	44 ± 6	6.728 ± 0.002	28 ± 1	0.946 ± 0.000	29 ± 2	3.97 ± 0.008
256	100	53 ± 8	4.437 ± 0.005	59 ± 8	9.425 ± 0.003	35 ± 8	1.069 ± 0.000	34 ± 7	4.20 ± 0.001
	1,000	53 ± 8	4.438 ± 0.007	59 ± 7	9.426 ± 0.002	35 ± 8	1.069 ± 0.000	34 ± 9	4.20 ± 0.000
512	100	83 ± 13	7.763 ± 0.003	92 ± 13	14.793 ± 0.004	59 ± 12	1.317 ± 0.000	58 ± 13	4.65 ± 0.000
	1,000	84 ± 11	7.765 ± 0.006	92 ± 11	14.796 ± 0.002	61 ± 11	1.317 ± 0.000	60 ± 10	4.65 ± 0.000
1024	100	122 ± 11	14.423 ± 0.002	134 ± 11	25.556 ± 0.003	90 ± 11	1.812 ± 0.000	89 ± 10	5.56 ± 0.001
	1,000	123 ± 3	14.424 ± 0.002	135 ± 4	25.555 ± 0.002	91 ± 4	1.812 ± 0.000	92 ± 8	5.56 ± 0.000
1514	100	170 ± 17	20.798 ± 0.001	172 ± 47	35.854 ± 0.003	130 ± 16	2.322 ± 0.064	129 ± 15	6.48 ± 0.000
	1,000	171 ± 6	20.796 ± 0.001	188 ± 9	35.853 ± 0.000	132 ± 6	2.317 ± 0.007	131 ± 9	6.48 ± 0.000

Table 4.1: Switch and loopback estimated OWD with different packet sizes and link speeds. Mean and standard deviation.

4.9 HwP100: VCU118 and Alpha Data ADM-PCIE-9V3

Hitherto, 1 and 10 Gbit/s active probes were described. In light of the extraordinary results of those probes, the natural step is to move forward in the link speed. Therefore, we have scaled the probe to support 100 Gbit/s. Even though, the current DPDK implementation supports a wide variety of 100 GbE NICs, the accuracy could not be compared against a hardware implementation. Thus, this probe is a unique element to accurate measure 100 Gbit/s infrastructure. We have built two probes using different boards with different part of the same FPGA family. The first one takes advantage of the Xilinx VCU118 development board (section 3.8.1), which includes two QSFP28 interfaces as well as one 1 Gbit/s Ethernet interface. This board is suitable for prototyping. On the other hand, the Alpha Data ADM-PCIE-9V3 is data center oriented board with two QSFP28 interfaces (section 3.8.1), the FPGA is three times smaller than the VCU118, however, for this design it is not an issue. For both board, the design was made from scratch. Even though they share most of the logic there are subtle differences regarding the infrastructure that connects with the physical parts. The focus will be on the Alpha Data ADM-PCIE-9V3 design.

In this case, we have designed a more complete active probe, see Figure 4.9. There are ARP and VLAN support, implemented using HLS. Each interface has its own IP and MAC address to fully compliant with network specifications. However, to avoid problem with timestamp synchronization of the probes, this design does not include any kind of synchronization scheme. Each probe has two interfaces, one to generate the packets and the other to receive and forward packets, acting as a packet reflector. In such a way that Round-Trip delay Time (RTT) is measured in the same probe that generates the packet avoiding synchronization issues across probes. Noteworthy, the timestamp resolution is as small as 3.013 ns.

The design uses two physical interfaces. **Interface zero**: is split into two independent designs. On the one hand, the transmitting side, a synthetic packet generator has been developed, using HDL to have full control of the logic. On the other hand, the receiving side is in charge of filtering the packets, analyzing them and generating a summary, implemented using HLS. **Interface one**: is targeted by the traffic generator, receives the packets and swaps the MAC, IP addresses and UDP ports in order to reflect the packet to the sender, implemented using HLS. In that way, the QoS parameters can be calculated, without synchronizing the probes. In what follows, the main components of the design are explained in detail.

4.9.1 Synthetic Packet Generator

This piece of hardware is written in Verilog and implemented as a Finite State Machine (FSM). In such a way that, the behavior is completely deterministic and it provides the most accurate measurements. This module is in charge of generating UDP packets that will carry useful information for the measurement. The packets can be generated at the maximum throughput, which is extremely close to the theoretical value in 100 Gigabit Ethernet links.

When a measurement is requested, some of the fields in the packets can be set, such as VLAN, source and destination IP addresses, source and destination ports, packet size, and Bit Error Rate Test (BERT) type used for the payload. Moreover, each UDP packet carries an `iperf` [93] compatible payload. Figure 4.10 shows the generated packet at IP level. Additionally, other configurable parameters are the amount of packets in the burst and the inter-packet gap between packets — number of idle cycles from one packet to another at the generation process. All those configurations are done through an interface from a program running in the computer hosting the FPGA card. The following BERT types have been implemented in the payload of the packets, so the active measurements can also be used to check if the optical modulation and the Forward Error Correction (FEC) implemented in the optical path are working properly:

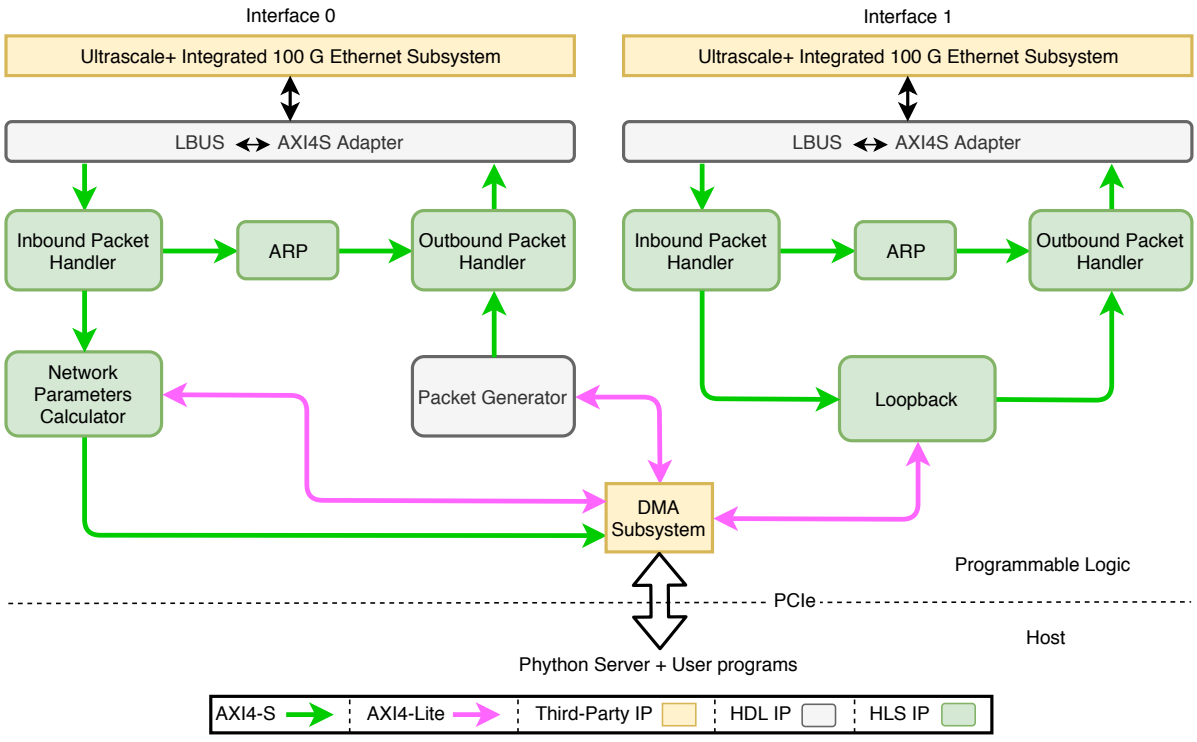


Figure 4.9: 100 Gbit/s active probe architecture overview.

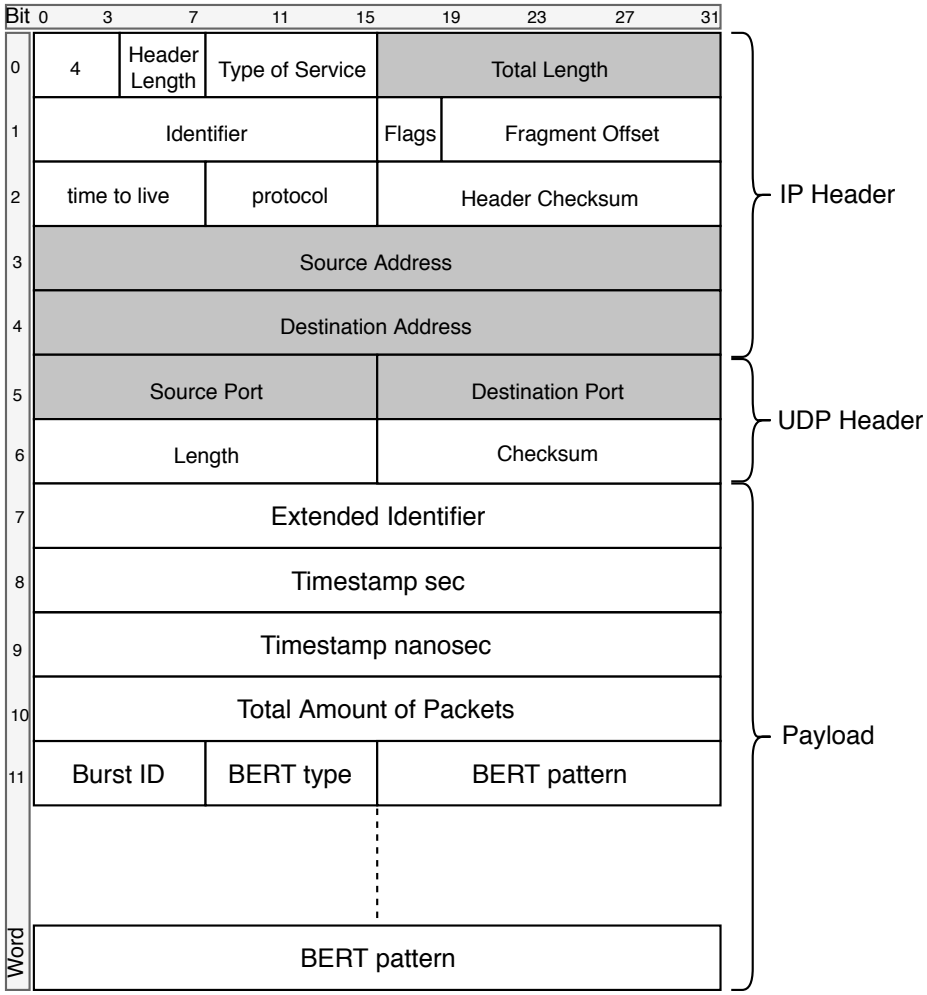


Figure 4.10: 100 Gbit/s synthetic packet structure.

- **PRBS** (Pseudo Random Binary Sequence): binary sequence that is difficult to predict and exhibits statistical behavior similar to a truly random sequence.
- **All zeros**: A sequence of all zeros.
- **All ones**: A sequence of all ones.
- **1:1**: A sequence composed of alternating ones and zeros.
- **1:7**: Also referred to as “1 in 8”. It has only a single one in an eight-bit repeating sequence.
- **2:8**: A pattern that contains a maximum of four consecutive zeros and only two ones.
- **3:24**: A pattern that contains the longest string of 15 consecutive zeros with only three ones.

4.9.2 Packet Filtering and Parameters Calculator

In the receiver side, packets are timestamped in the moment they reach the FPGA side. The packet handler module is in charge of filtering packets by type. For instance, ARP and UDP packets are taken into account in this implementation. After that, UDP packets are parsed and those fields that are configurable are verified in order to check if the packets match with the required configuration. If so, the useful information is extracted to compute the packet train parameters. This information is fed to the statistics generator — a piece of hardware very similar to the Network Parameters described in section 4.7.1— which collects it. The result of the measure is then sent to the HOST machine through PCIe. However, for the VCU118 implementation an IPFIX report is generated and sent through the 1 Gbit/s interface. All the modules described in this subsection have been implementing using HLS.

4.9.3 Evaluation Testbeds

We have used the same two scenarios as described in section 4.8.1. The first one, the probe connected in a loopback fashion. The second scenario, in this case we have used two switches: Huawei Cloud Engine 8800 and Mellanox MSN 2100-CB2FC. Both switches were configured to support hybrid interfaces. In such a way that, the interfaces are able to handle packets with VLAN and without VLAN.

All the experiments featured packet trains of 1,000,000 packets with frame sizes of 64-Byte to 1514-Byte with a step of 1-Byte, excluding frame preamble and check sequence. Experiments using packets with VLAN and without VLAN were carried out. Each experiment was repeated ten times to obtain mean and standard deviation for throughput and RTT. This probe aims at testing the Metro-Haul infrastructure where a reliability of six nines is sought, therefore at least a million of packets are needed.

4.9.4 Experimental Results

The throughput measurement for the experiments without VLAN are shown in Figure 4.11. Whereas, Figure 4.12 shows the throughput results when VLAN is used. At a glance, it seems that the results are really close to the theoretical. Yet, a closer look Figure 4.13 and Figure 4.14 shows that there is little difference with the theoretical value. However, the almost 0.02% difference is negligible. In this case, we have not carried out experiment with software-based solution because of the poor result of such solution at 10 Gbit/s.

Figure 4.15 shows the RTT measurements for the different scenarios. The results when VLAN is used are almost identical to the results when there is no VLAN. It is

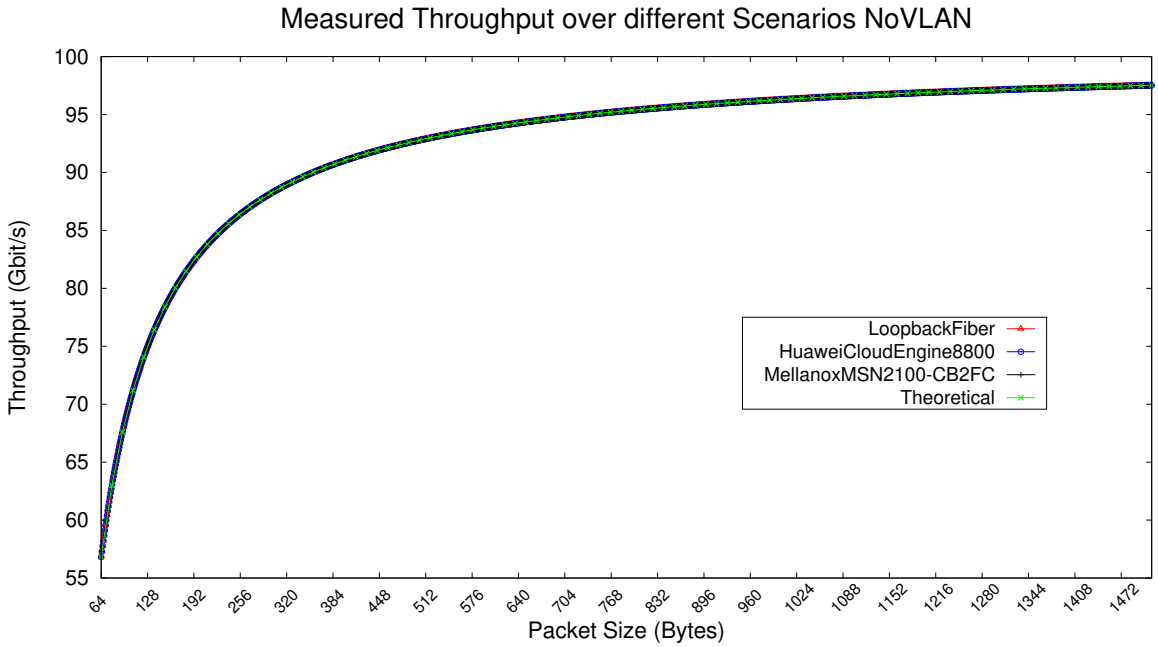


Figure 4.11: 100 GbE active probe measured throughput when NoVLAN.

also noteworthy that the Mellanox switch is 300 ns faster that the Huawei one. In this case, we do not need to compute a function to calibrate the system. Since we have every possible value, the calibration is done subtracting the measure value with the correspondent loopback value. We believe that the drop in latency around 832-Byte is due to a mode change in the Mellanox switch.

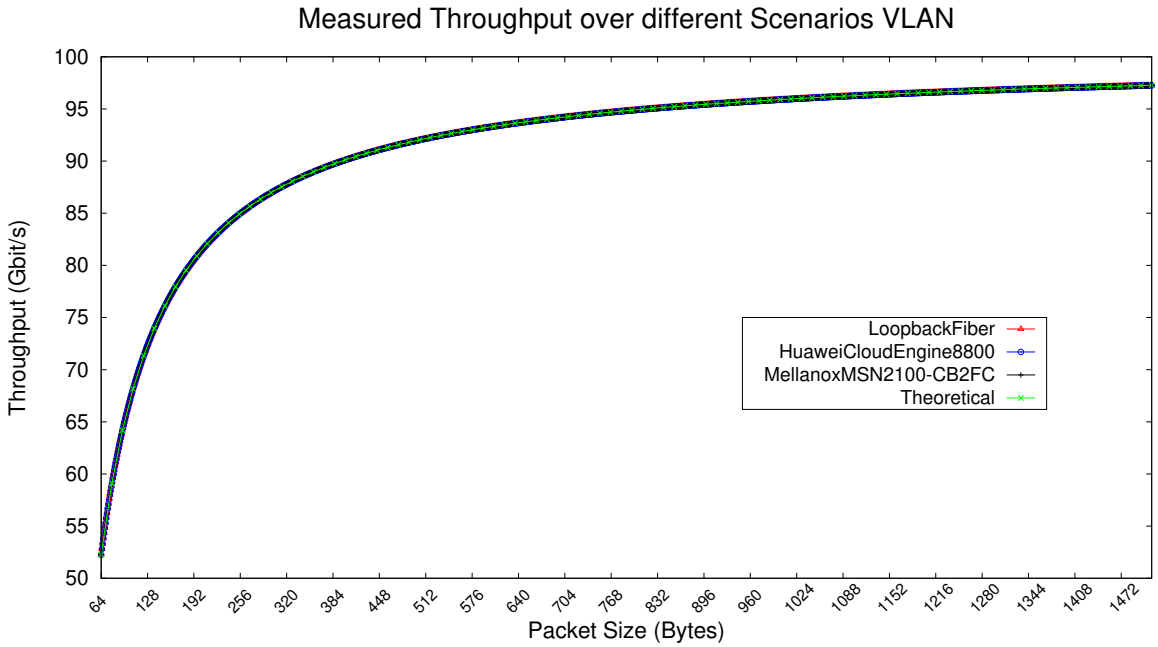


Figure 4.12: 100 GbE active probe measured throughput when VLAN.

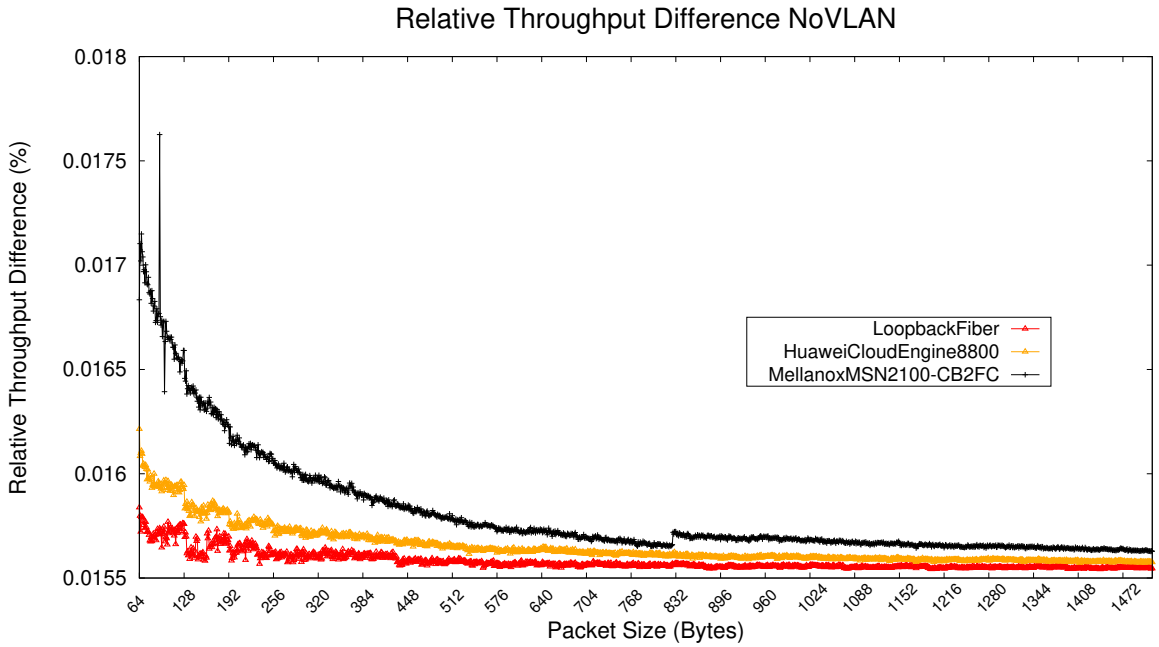


Figure 4.13: Error of 100 GbE active probe measured throughput when NoVLAN.

At the same time that the RTT experiments were carried out, we measured the jitter of our solution for the different scenarios. The results are shown in Figure 4.16. The jitter in all experiment is negligible compared to the latency, again this fact confirms that FPGAs are the solution to perform active measurements.

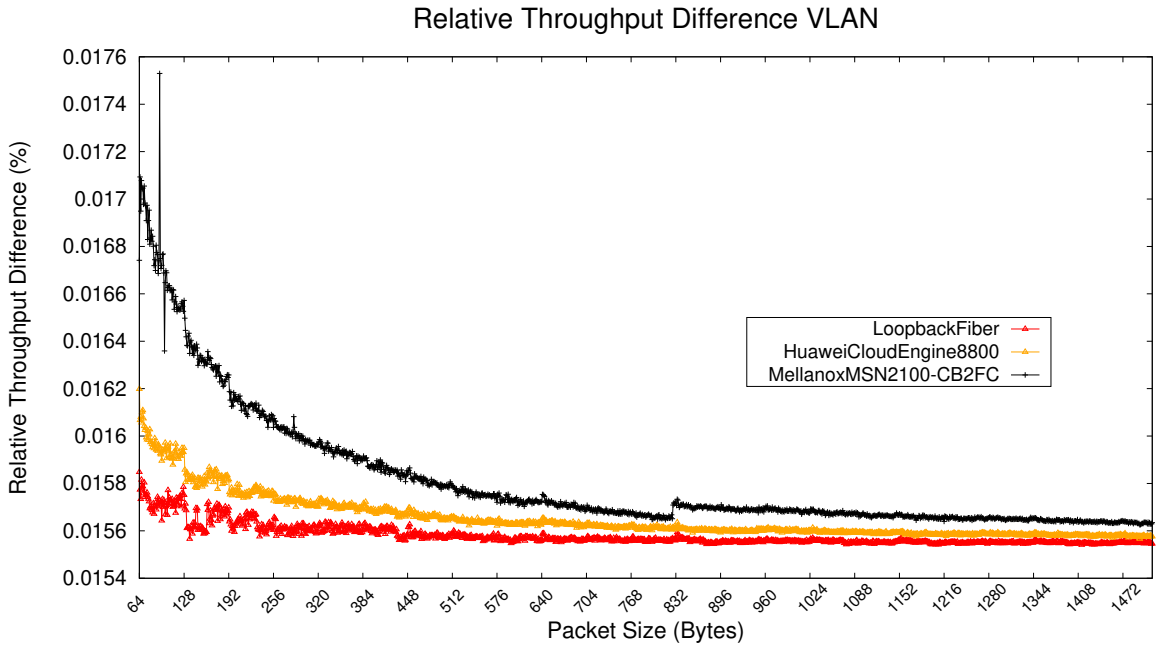


Figure 4.14: Error of 100 GbE active probe measured throughput when VLAN.

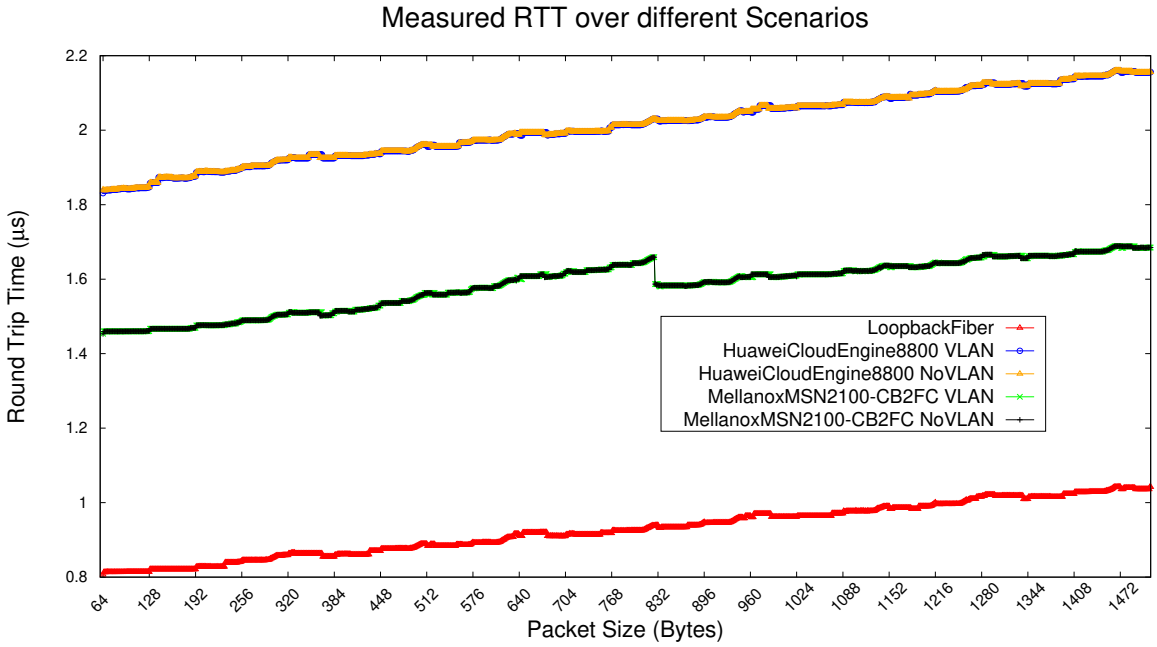


Figure 4.15: Round trip time measurement at 100 Gbit/s for different devices.

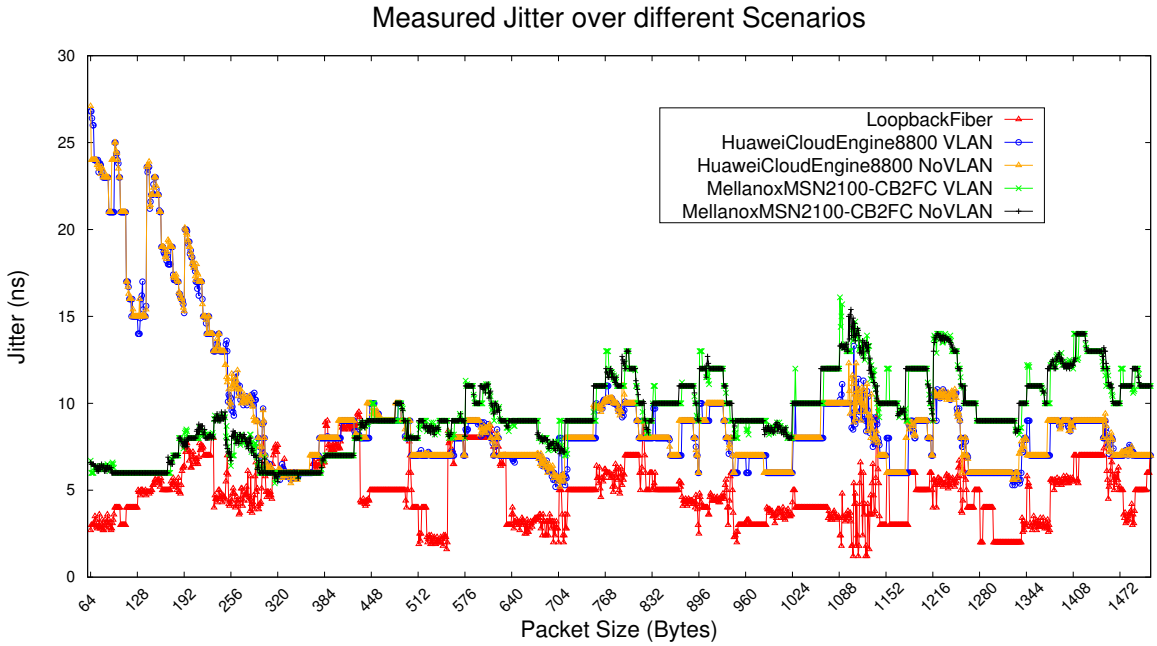


Figure 4.16: Jitter measurement at 100 Gbit/s for different devices under test.

4.10 Conclusions and Future Directions

The increasing speed of communication networks poses a serious challenge for testing and calls for more accurate probes. In this chapter we presented the advantages of FPGA technology to implement accurate and affordable testing appliances for high-speed networks. Although software-based solutions are undoubtedly the most convenient approach in terms of deployment and development costs, it has been shown that the non-determinism of software severely limits the accuracy of such solutions at multi-gigabit-per-second speeds. Moreover, non-determinism arises in the NIC itself and its connection to the system (PCIe, chipset), so using GP-GPU accelerators or multi/many core architectures does not help to solve such an issue.

In order to assess the benefits of FPGA active probes for high-speed networks, several solutions have been proposed, ranging from plain programmable logic to SoC devices. The first one runs at 1 Gbit/s and is based on the ZedBoard. The second one runs at 10 Gbit/s on top of NetFPGA-10G. The third one runs at 100 Gbit/s on a Virtex Ultrascale+ device. The packet-train technique has been chosen to perform the measurements due to its well-known features of accuracy, interference immunity and low network overhead during testing. With the help of these three proof-of-concept designs, the advantages of hardware solutions have been exposed in terms of determinism and accuracy when compared to the software alternative. Another contribution is to present the benefits of open-source platforms and high-level synthesis for improving FPGA design productivity. In this context, most of the development of the proof-of-concepts has been done using HLS, apart from the packet generator, which was done using HDL to have full determinism. This methodology has proven to reduce the development cycle without a degradation on the results.

Table 4.2 provides a qualitative and quantitative summary of FPGA versus software solutions for HwP1, HwP10 and HwP100. As it was evidenced in Figure 4.5, software solutions only provide a good accuracy for throughput measurements at 1 Gbit/s and when using large packet sizes. At 10 Gbit/s the software accuracy is very poor, even if using kernel-level approaches such as the one evaluated in this thesis. In terms of designability and design time, the use of HLS brought those variables in the FPGA design closer to the software standard. Regarding costs and power consumption, the results presented in Table 4.2 are the corresponding to the prototypes that have been evaluated. For the software solution, a high-end server with a 10 GbE card was used. For HwP1, the configured system includes the ZedBoard card (section 3.6), a GPS receiver and a GbE FMC daughter card. For HwP10, a NetFPGA-10G (section 3.7) card was used (academic price), along with a low-end computer attached to it. Finally, for HwP100 an ADM-PCIE-9V3 card (section 3.8.1) and a low-end server attached to it were used.

Finally, in this chapter the scalability of the packet-train technique fitting different network speed has been shown. What is more, the use of dedicate hardware has also been proven, especially when considering the nanosecond-accuracy. Therefore, this technique could be applied to the upcoming 200 and 400 Gbit/s, for instance, using a VCU129 board [94]. Nowadays, such very high-speed devices are very expensive, however there is no COTS counterpart able to handle such speed. Therefore, in such state-of-the-art technologies using FPGA becomes mandatory. Locally, in this chapter we have demonstrated that using HLS the development phase is simpler and shorter.

Feature	SW	HwP1	HwP10	HwP100
Approximate cost	5000 USD	1400 USD	3500 USD	9000 USD
Power consumption	160 W	8.5 W	120 W	150 W
BRAM Blocks (BRAM36)	N/A	86 out of 140	11 out of 324	148 out of 720
DSP Blocks (DSP48E)	N/A	0 out of 220	2 out of 96	17 out of 2280
Number of Slices	N/A	8,210 out of 13,300	28,996 out of 37,440	Not Reported
Slice LUTs	N/A	22,224 out of 53,200	79,099 out of 149,760	121,606 out of 394,080
Slice Registers	N/A	24,589 out of 106,400	81,646 out of 149,760	207,198 out of 788,160
Timestamp resolution	10 μs	10 ns	6.4 ns	3.103 ns
Designability	Easy	Moderate	Moderate	Advance
Design time	Weeks	Months	Months	Months
Engineer skills needed	Drivers	FPGA	FPGA	FPGA
Maximum rate supported	10 Gbit/s	1 Gbit/s	10 Gbit/s	100 Gbit/s
VLAN Support	✓	✗	✗	✓
Measure throughput 1 Gbit/s	poor	✓	✓	✓
Measure throughput 10 Gbit/s	poor [86]	N/A	✓	✓
Measure throughput 100 Gbit/s	not verified	N/A	✓	✓
Measure OWD 1 Gbit/s	✗	✓	✓	✗(only RTT)
Measure OWD 10 Gbit/s	✗	N/A	✓	✗(only RTT)
Measure RTT 100 Gbit/s	✗	N/A	N/A	✓

Table 4.2: Features summary of software and hardware prototypes.

CHAPTER 5

PASSIVE MONITORING

In this chapter we explore the FPGA capabilities to implement two passive bump-in-the-wire implementation which aim for reducing the input traffic load smartly. The idea is to plug this appliance between the point to monitor and the traditional software-based probe, in such a way that the FPGA is in charge of preprocessing the traffic and reducing the output traffic. Thus reducing the computing performance needed as well as the reducing the storage system of the software probe.

5.1 Introduction

Passive monitoring, which is based on capturing traffic for real-time or for offline forensic analyses, is widely used to evaluate the healthiness of networks (section 3.2.1). Over the years many tools to perform this type of analysis have been developed. What is more, among GNU/Linux tools we can find `wireshart, tcpdump, tshark`. However, network traffic monitoring is becoming increasingly challenging to manage due to the relentless growing speed of network links. This effect is even more noticeable at 100 Gbit/s; the huge volume of data makes it very difficult to perform online analyses or event to store traffic for subsequent forensic investigations. Researchers have studied different architecture to be able to cope with such speeds, however the solutions do not completely address the matter.

In this context, it is mandatory to implement mechanisms to significantly reduce the amount of information used in the analysis without losing any relevant details, thus, it

is therefore mandatory to carry out some sort of smart filtering and/or capping in the network traffic to be analyzed. What is more, network traffic monitoring usually faces the problem of packet duplication, which arises when port mirroring is being used. That is, when traffic is copied from the ports of a switch or a router that are being monitored, to a mirror port where a monitoring probe is attached. Thus, a packet can be copied twice, both at the ingress and egress ports, therefore generating duplicates. Information redundancy caused by packet duplication not only leads to increased workloads at the monitoring probes, but also calls for more disk space to store the network traces. Actually, packet duplication might increase 100% the monitoring load, in the worst case.

In this chapter, we present two FPGA-based bump-in-the-wire passive probes which aim for reducing the traffic load in the software probe. We present two different approaches: a) smart capping cypher packets and b) removing duplicate packet. These might seem straightforward ideas, however, they have been proven to alleviate the traffic load in the probe. What is more, they can even be combined together to further reduce the traffic. Hence, extending the usability of traditional monitoring tools.

To do so, we have taken advantage of the Ultrascale(+) architecture of the VCU108/118 (section 3.8.1) boards and High-Level Synthesis (HLS) (section 3.3.2) to cut down development time. However, we also used a traditional approach based on Hardware Description Language (HDL) to optimize critical areas of the designs. Both alternatives are a proof-of-concept based on a store and forward architecture, where the packet is stored until a decision is made, therefore, the packets are forwarded, capped or dropped. Moreover, we have achieved designs able to work at 100 Gbit/s, consequently, reducing the costs and computational load of the host associated to the network analysis.

5.2 Packet Capturing: Related Work

With respect to traffic capture, the authors in [95] studied the hardware that is necessary to store 10 Gbit/s network traffic at line rate. Even with today's equipment the challenge remains, the work [96] presents the challenges to capture and store at 40 Gbit/s, not only capturing is difficult, but also, storing the data for further analysis becomes demanding. Trying to scale up the problem at 100 Gbit/s makes the task extremely difficult and costly. The author in FlowScope [97] presents an event triggered packet capture, they claim to be able to achieve more than 120 Gbit/s capture rate, but, the storage is done when a trigger happens. The literature shows significant progress in capturing and storing traffic at high-speed rate, however the hard disk drive capacity is limited and expensive at such speed. Consequently, the need of reducing the traffic to be analyzed and eventually stored is paramount, thus saving compute resources and

capacity on the hard disk drive.

5.3 Capping Cypher Packets

The ratio of encrypted traffic is relentlessly increasing, and therefore the information that can be extracted from the packet payload is limited. For such encrypted traffic, storing the payload is most times useless and has a counterproductive effect in the storage. For that reason, we use a method able to identify plain text (that is, human readable) in the network packet payload. The method is based on both detecting bursts of printable American Standard Code for Information Interchange (ASCII) characters and computing the percentage of such printable characters in the packet payload. This method has proven to be very effective in reducing the amount of information used in traffic analysis, by saving only the headers of packets with encrypted payloads. Needless to say, such implementation helps the current tools to handle a smaller amount of traffic than if they were connected directly to the network.

In this section we explore a bump-in-the-wire FPGA implementation of such method, able to identify plain text (human readable) in the packet payload, which is extremely useful for forensic and real-time analyses. As stated in section 3.8 in the board at hand, packets are received through a 512-bit (64-Byte) AXI4-Stream interface at 322.265625 MHz (3.103 clock period), and the maximum packet rate is 148.8 million packets per second. At this rate, it becomes a serious challenge to implement the selected algorithm for discriminating ciphered traffic. In order to address this challenge, a low-level, handmade architecture was developed using VHDL and integrated in a HLS flow in order to reduce the development time.

The approach to recognize encrypted traffic and capping it, since it is mostly irrelevant for further analysis, has already been explored. There are a few works in the literature with different approaches to identify encrypted traffic; in the following paragraphs we summarize them.

Velan *et al.* [98] preset a survey of methods to detect encrypted traffic. The work [99] presents a survey in traffic classification based on the payload. In the paper [100] the future directions of traffic classification are studied. In [101] the authors present a method to detect encrypted traffic in real-time, evaluating only the first packet on the flow. In such work, 94 % of encrypted traffic is detected as encrypted. Regrettably, that solution is impracticable onto FPGA at 100 Gbit/s because it needs a large memory to save the information of flows —on-chip memory is a limited resource on FPGAs—, even with an efficient implementation of a hash table, because the collisions will reduce the performance. Similar to the previous work, in [102] a mechanism is implemented to

recognize applications encapsulated in Secure Sockets Layer (SSL) connections, which is based on the size of the first packet in the connection. Such method achieved more than 85 % accuracy, but it needs again a large memory to identify the flows, which makes it less suitable for a hardware implementation. In [103] an FPGA-based solution is implemented that achieves 786,432 concurrent flows using an external QDR-II memory at 10 Gbit/s, a small number of flows compared with the amount of concurrent flows in a real core network. Ongoing research try to tackle flow-based monitoring at 100 Gbit/s on FPGAs one such example is [104], however, the number of supported flows is negligible compared to the actual concurrent flows in a 100 Gbit/s links. Therefore, we discarded the idea of using flow information to detect cyphered packets owing to the huge memory consumption, given that in other software-based works [105] authors deal with about 10 Million of flows at 10 Gbit/s. However, the new FPGA devices with High Bandwidth Memory (HBM) could open up new architectural design able to deal with flow-based monitoring. Additionally, in [106] a hybrid method to identify encrypted application traffic is presented, combining a signature-based method and a statistical analysis method. Such hybrid method achieves a classification accuracy above 99%. But that work does not consider the speed at which they can classify the traffic, and unfortunately it seems to be impossible to achieve 100 Gbit/s with FPGAs using this method. The patent [107] also describes an encrypted-traffic discrimination device, but the description about the architecture is very vague and generalist. In fact, the speed that can be achieved by the proposed system is not provided. On the other hand, we can find in the literature traffic classification using stateful machine learning implementations [108, 109]. These machine learning algorithms are very expensive to implement onto FPGAs, and what is more, a previous phase is necessary to train the network.

The literature review, previous paragraph, reveals that the research on detecting encrypted traffic is based on inspecting flows, such stateful algorithms are not very well suited for FPGA design due to the huge amount of memory needed. However, Uceda *et al.* [110, 111] in 2015 introduced a stateless algorithm able to detect cyphered packets. These works use bursts of consecutive printable ASCII characters and the percentage of printable ASCII present in a packet as the basic metrics to recognize relevant packets that need to be further analyzed (*i.e.* not encrypted packets). The output of this algorithm can be a whole packet without encrypted traffic or a capped packet with just protocol headers (*e.g.* IP or TCP) for encrypted traffic. For example, this implementation can keep some content of the handshake of SSL flows such as the X.509 certificate, because of the amount of ASCII text in the payload, and discard the payload in the rest of the packets of the encrypted flow. Given the good results provided in the original work, we exploit the stateless characteristics of this method and leverage the massive parallelism of FPGA as well as HLS to make an architecture able to filter encrypted network traffic at 100 Gbit/s.

Given the previous efforts on encrypted traffic discrimination found in the literature. The main contributions and distinguishing features of our work can be summarized as:

- This architecture is able to deal with fully loaded 100 Gbit/s Ethernet links, guaranteeing that the complete payload of packets can be processed at line rate.
- The presented approach is able to be extended to filter by other different criteria, while maintaining the line rate capability.
- The work shows the benefits of using a mixed approach combining HLS and low level design for critical parts, which reduces development effort and increases performance.

5.3.1 Method to Identify Relevant Packets

It is estimated that in 2016, over 70 % of the traffic was encrypted [112]. However, the 30% remaining continue using plain-text in the payload. In this part of the passive monitoring we focus on finding non-encrypted traffic, in this case packets that carry information in plain text, specifically those encoded in ASCII and the subset of variable 8-bit Unicode Transformation Format (UTF-8). According to their payload, different types of packets can be defined [110] (Figure 5.1):

(a) Completely binary (encrypted). Class I.

(b) With printable ASCII at the beginning of the payload. Class II.

(c) Mix of printable ASCII and binary. Class III, IV and V.

(d) Pure plain-text ASCII. Class VI.

In the works [110, 111] the authors studied statistically the relation of consecutive printable ASCII characters and the percentage of printable ASCII characters in the payload to determine the usefulness of the payload of a given packet for the network analyst. Additionally, they proposed two software-based solutions: the first one inspected the whole packet more accurate but not able to achieve 10 Gbit/s, and the second one implemented an algorithm able to achieve line rate at 10 Gbit/s but at the expense of not scanning the whole payload, thus less accurate.

In this implementation, based on such previous works, the speed link was scaled to 100 Gbit/s and furthermore, all bytes in the payload are inspected in order to increment the granularity and accuracy in the decision. Noticeable, our implementation does not trade accuracy for speed. Moreover, this implementation uses fixed parameters to

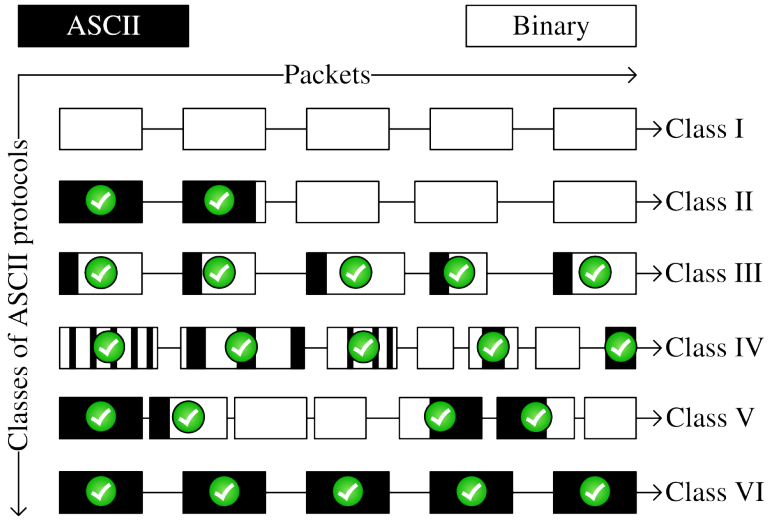


Figure 5.1: Taxonomy of classes of protocols according to how printable ASCII data is carried. Source: [110].

simplify the design: the packet is accepted as non-encrypted when a burst of twelve consecutive printable ASCII characters is detected or the percentage of printable ASCII is greater or equal than 50% of the packet size. If the packet does not meet these rules, it is considered binary and then, just the first 64-Byte are kept (headers containing information of Ethernet, IP, and transport protocols), which is the only part of the packet that will be useful for a network analyst.

5.3.2 Architecture

The basic structure to filter encrypted traffic is the so-called *Analyzer Unit*, which is a store and forward architecture, depicted in Figure 5.2. The architecture is composed of three main blocks that will be explained later on. The first version was described in C/C++ using Vivado-HLS tool [68], and using directives as described in [35, 113]. The proposed architecture can be easily extended to filtering any kind of traffic by simply modifying the *decider* module, as long as the filtering criteria are stateless or the output First-In First-Out (FIFO) does not get full. One of the biggest challenges is to obtain a design which meets the 3.103 ns of clock period.

Receiver: The receiver is connected to the incoming 100 Gbit/s Ethernet interface (source of traffic to be filtered) through a 512-bit (64-Byte) width AXI4-Stream interface. Its function is to receive Ethernet traffic and split it into two streams: (a) one is an exact copy of the input, which is sent to a FIFO to be stored until the decision is made. (b) A byte-by-byte reduction to a 64-bit boolean vector that represents if the corresponding byte is a printable character (byte decimal value between 32 and 126) or not. This vector

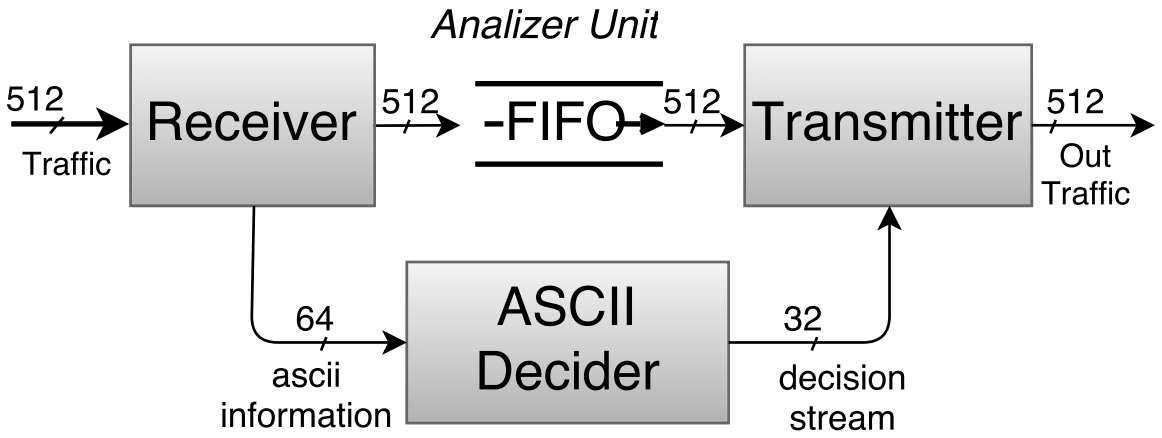


Figure 5.2: Analyzer unit, which is a store and forward architecture.

will be called from now on the *vector of ASCII information*, which is later sent to the decider to be analyzed. The reduction from 512-bit to 64-bit is a naïve comparison, which can be made even combinational. It corresponds to detecting for each octet if it is between the hexadecimal values 0x20 and 0x7E, both inclusive. A simplification is to consider also `DEL` (0x7F) as printable ASCII. If so, the comparative is even simpler: It is only needed to check the three most significant bits of each byte. This module was developed in C using Vivado-HLS.

Transmitter: This module is in charge of generating the output stream. It receives two input streams: The *decision stream* indicates the amount of bytes to be transferred, and it comes from the *ASCII decider* module through a 32-bit width AXI4-Stream. The other stream is coming from the FIFO and contains the already analyzed packet. When a valid decision arrives, the logic starts to read data from the FIFO and forwards the amount of bytes that the decision informed. If the decision has less bytes than the packet, the rest of data are discarded in order to empty the FIFO for the next computation. This module was developed in C using Vivado-HLS. This architecture can also be used to discard packets, when the decision is zero, the whole packet is read but not forwarded.

ASCII Decider: The ASCII decider receives the 64-bit *vector of ASCII information* indicating which characters are printable. This core uses that information to count the amount of printable characters and to detect sequences as explained in section 5.3.1. At each clock cycle, two operations should be done: i) the 64-bit vector is reduced to a 7-bit number that represents the amount of printable characters in the current transaction; and ii) using the last 11-bit from previous transaction it tries to find sequences of twelve consecutive printable characters.

Initially, we implemented this module using HLS. However, the implementation had an initiation interval bigger than 5 clock cycles. Therefore, we could not handle scenarios

with short packets. Consequently, we decided to implement this module using HDL to use low level elements efficiently.

Verifying the consecutive number of printable ASCII is computationally straightforward; it implies to test 64 times in parallel the presence of twelve consecutive '1'. In a Xilinx UltraScale device, it implies 44 LUTs and less than 2 ns using a behavioral VHDL description. Nevertheless, counting the amount of '1' is a much harder problem, and it is deeply analyzed in section 5.3.3.

Finally, in order to take the decision of how many bytes are to be transferred, a 4-state Finite State Machine (FSM) as shown in Figure 5.3 is implemented. Each state and the transition between states are described below.

- *WAIT_FIRST_WORD*: This is the first state after reset. When the packet has 64-Byte or less, the packet must be sent completely, then decision is *64*, and next state is *WAIT_READY*, otherwise the packet is greater than 64-Byte and the next state is *COUNT*. While there is not any packet, state does not change.

- *COUNT*: This state sums printable ASCII characters and bytes received. When a sequence of twelve printable ASCII characters is detected, the packet must be sent completely; otherwise, if the packet ends, the amount of printable ASCII characters is compared with the amount of received bytes: if it is greater or equal than 50 % the packet must be sent completely, otherwise only the first 64 bytes are sent, then the next state is *WAIT_READY*. If no sequence or packet ends are detected, the state does not change.

- *WAIT_READY*: This state waits for handshake (assert ready signal) from transmitter. When ready is valid and packet ends the next state is *WAIT_FIRST_WORD*, otherwise if the packet has not ended yet, the next state is *WAIT_PACKET_END*. If ready is not asserted, the value on decision does not change and the state does not change, what is more, decider ready is set to '0'.

- *WAIT_PACKET_END*: If we arrive to this state a sequence has been detected and the decision is sent, however, the packet has not ended yet, then this state waits until signal last or pkt_end is asserted to change the state to *WAIT_FIRST_WORD*, otherwise the state does not change.

5.3.3 Counting the Amount of ones in a Vector

The problem to count the printable ASCII characters in a 512-bit transaction is reduced to the problem of counting the amount of '1' present in a 64-bit *vector of ASCII*

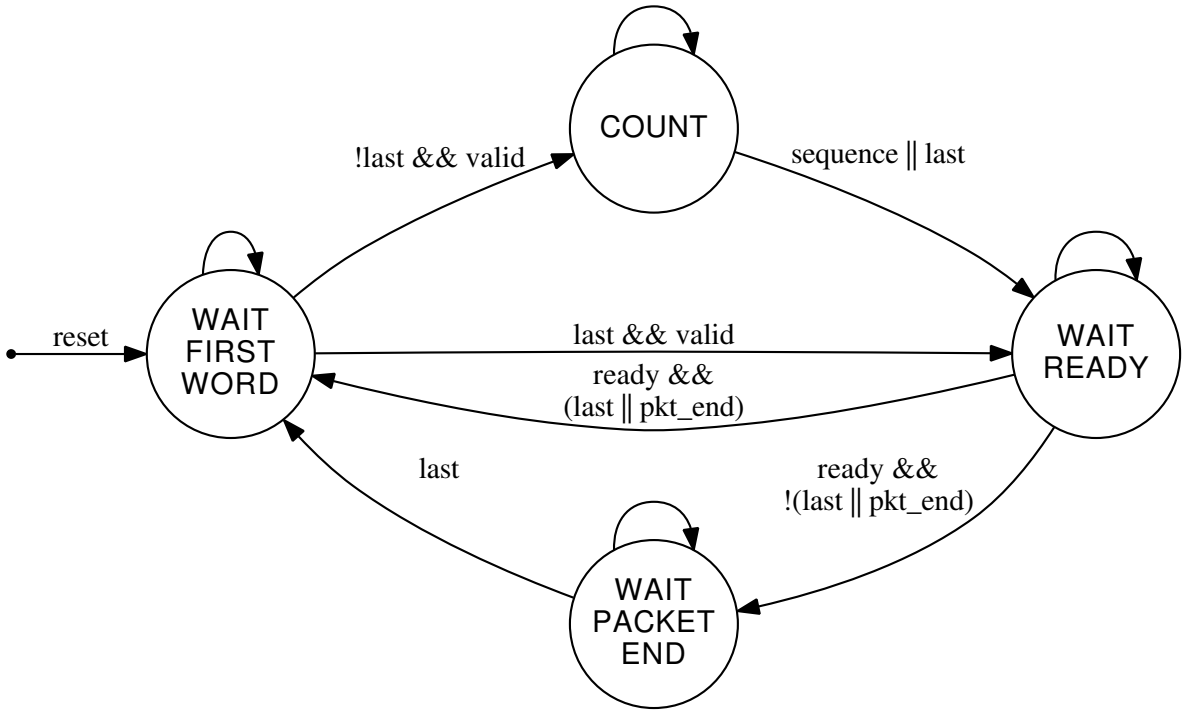


Figure 5.3: ASCII decider finite state machine implementation.

information. Such a task sometimes is called Hamming Weight or popcount function. Sklyarov *et al.* [114] studied different implementations of Hamming Weight and compared their results with other implementations, however, with a vector of width 64 the latency is 5.2 ns, almost doubling the clock period required for this architecture. Consequently, to solve the problem different alternatives were studied to achieve the necessary timing.

A naïve implementation in VHDL gives poor results, then we have studied different alternatives that fit better in a 6-LUTs architecture present in Xilinx devices. The idea is to use of n-to-k reducers (or counters), also a sort of Carry Save Adder (CSA) that counts n-bit giving k-bit results, see section 6.6.4 for more information about CSA. We have evaluated different alternatives:

- **Using 7-to-3 reduction trees (V1)**: The main building block is a 7 to 3 reducer since it can be efficiently implemented using three times two 6-LUTs and a muxF7. Starting with nine 7-to-3 reducers — the result of such reductions are S(0) to S(8) in Figure 5.4 — as result nine 3-bit numbers plus one bit (Figure 5.4.A) to obtain 3-bit results. Then we reduce again as shown in Figure 5.4.A obtaining three 3-bit number plus a 4-bit number that can be added using a ripple carry adder tree.

- **Using 7-to-3 and 8-to-4 reduction tree (V2)**: In this approach, after the first 7 to 3 reduction we apply an 8 to 4 reduction (using four 6-LUTs, two muxF7 and a muxF8 per bit) with the aim to reduce the logic depth (Figure 5.4.B). This

approach increases area, and worsens the delay (due to more fan-out and network congestion).

- **Using reduction trees 6-to-3 (V3)**: Since a 6 to 3 reducer can fit in 3 parallel 6-LUTs we expect to reduce net congestion and improve delay. The first state reduces from 64-bit to eleven 3-bit numbers — T(0) to T(10). Then, again a second level reduces to six 3-bit numbers and a third level to three 4-bit that can be added with a three input ripple carry adder (Figure 5.4.C).

For the two first proposed architectures (V1 and V2), a first step reduces the 64-bit vector to nine 3-bit numbers plus an additional bit that should be added later. The resulting dot graphic to be added is depicted at Figure 5.4.A and Figure 5.4.B respectively for V1 and V2 architectures. In case of V1 a second level composed by three parallel 7-to-3 reducers and *ad-hoc* reducer that outputs 4-bit as shown at Figure 5.4. A produces three 3-bit and a 4-bit number that are added by an addition tree. On the other hand,

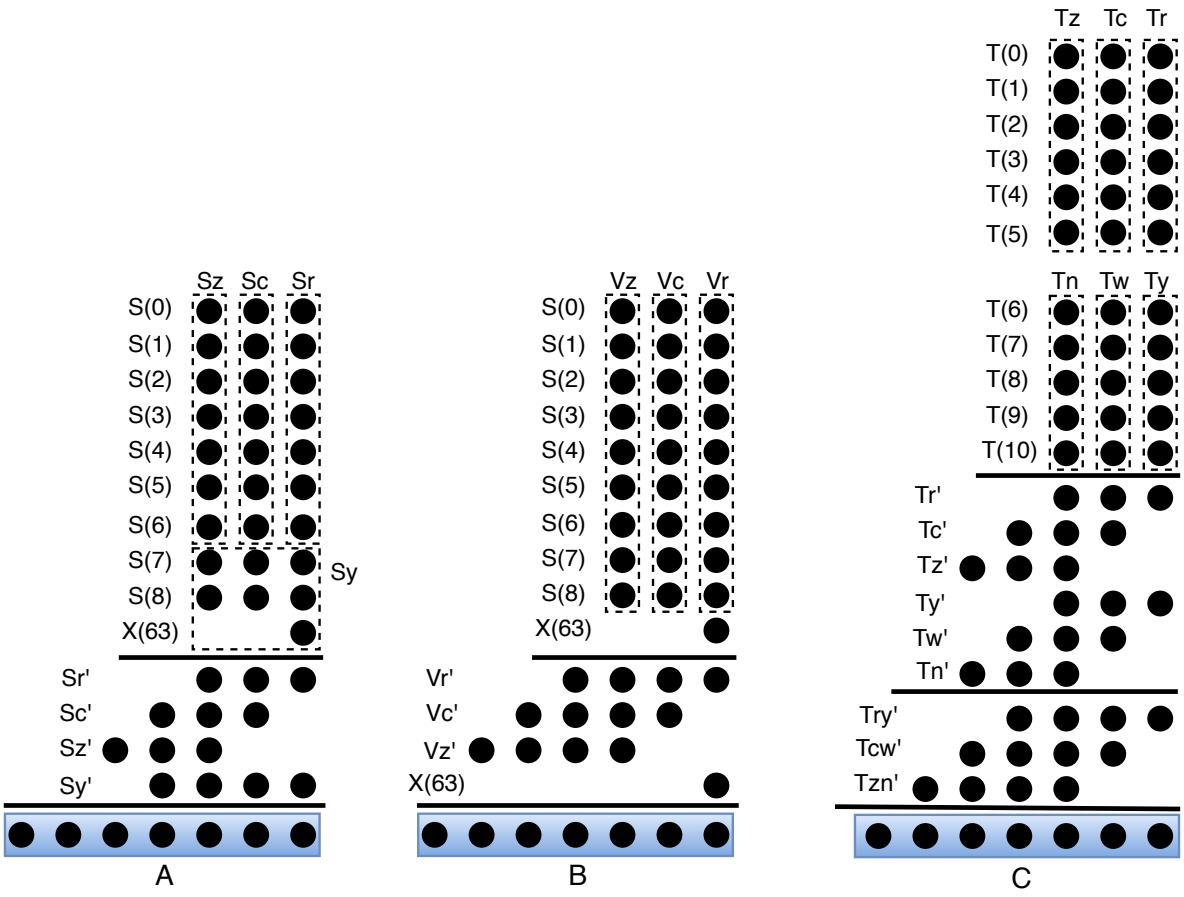


Figure 5.4: Dot graph for different reduction trees: A) two levels of 7-3 reduction and finally quaternary adder. B) 7-3 and 8-4 reduction and finally ternary adder. C) two levels of 6-3 reduction and a level of two input adders and a final ternary adder.

Version	Critical Path (ns)	LUTs
naïve	2.99	93
7-to-3 (V1)	2.89	92
7-to3 & 8-to-4 (V2)	2.98	80
6-to-3 (V3)	2.65	65

Table 5.1: Critical path and resources utilization for different tree reduction implemented on a XCVU095-2FFVA2104E FPGA.

V2 uses three parallel 8-to-4 reducers as proposed in Figure 5.4.B, generating a number than can be added using ternary adders. Finally, Figure 5.4.C shows for V3 the first step of 6-to-3 reducer, followed by the second level of reduction, composed by a three × 6-to-3 reduction and three × 5-to-3 reduction tree. The third logic level finally reduces to three 4-bit numbers to be added by a ternary adder.

The implementation details of these alternatives for a XCVU095-2FFVA2104E FPGA are summarized in Table 5.1, where the 6-to-3 reduction clearly gives the best results both in area and delay.

5.3.4 Implementation Results

The implementation targeted the Xilinx VCU108 board (section 3.8.1), and was made using the Vivado Design Suite 2016.2. The implementation was simulated with the Integrated Block for 100G Ethernet [115] obtaining successful results, achieving line rate. The proof-of-concept was also evaluated in a real implementation feeding the design with different pcap traces. The clock timing constraint of a 3.103 ns period (322.265625 MHz) is satisfied and the resource usage for a Xilinx Virtex UltraScale XCVU095-FFVA2104-2-E device is provided in Table 5.2. It should be noted the small footprint of the design, which uses less than 4% of the total area. Thus, a smaller and cheaper device such as an XCVU065 could be used in a commercial implementation of the design, or more computation can be included in the same chip.

Resource	used	max available	% usage
LUT	11761	537,600	2.19
FF	31016	1,075,200	2.88
BRAM	53	1,728	3.04

Table 5.2: Resource usage in a Xilinx UltraScale xcvu095.

5.4 Packet Deduplication

Packet duplication has usually been an undesirable side-effect of traffic monitoring. Packet duplication produces redundant information in the monitoring probe, which leads to an increased workload, and calls for more disk space to store the network traces.

The most common way to capture traffic from a network is using the port mirroring feature of the switching devices, which is also known as Switched Port Analyzer (SPAN). This feature, included in most enterprise-grade switches and routers, consists in making a copy to a mirror port of packets traversing monitored ports. The mirror port is then connected to a monitoring probe running a capture traffic engine such as `tcpdump`, `tshark`, DPDK-based tool, etc. Port mirroring unavoidably creates packet duplication. For instance, in Figure 5.5, when computer A sends data to computer B, packets pass through port 1 (ingress port) and port 2 (egress port). If both port 1 and port 2 are configured to be mirrored, the packet appears twice in the mirroring port. In the worst scenario, all traffic could be duplicated, which implies a large resource wasting. What is more, it is a challenge to know the interval between copies of the same packet, because such lapse of time is function of switching time, queuing delay and traffic load at given time. Such interval is a key metric for a deduplication device, since, it defines time window where a duplicate can be found.

Packet deduplication has had little relevance in the academia, because off-the-shelf hardware could, in the past, deal with this problem. However nowadays, with the advent of 100 Gbit/s networks, this problem is becoming increasingly challenging, since time between packets could be as small as 6.7 ns. Thus, packet deduplication becomes key for monitoring 100 Gbit/s networks, because at such speeds, processing unnecessary

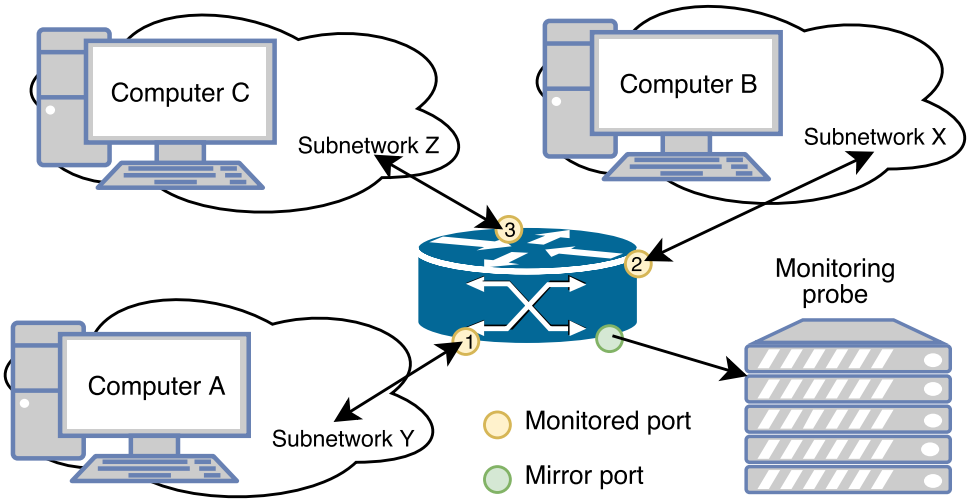


Figure 5.5: Simple monitoring set up.

information causes a penalty that cannot be afforded. Additionally, packet deduplication brings additional benefits, as authors in [116] discuss, for instance reducing the amount of stored data on disk.

In this part of passive monitoring, we have explored FPGAs to detect and discard duplicate packets caused by port mirroring at 100 Gbit/s speeds. A novel architecture is introduced to detect and remove duplicated packets, which is based on using an element-based (not time-based) sliding window. For this, a hash function is used to represents each packet with a k-bit key. There is a total of N sliding windows, which are addressed using $n = log_2 N$ bits of the key, and each window contains M elements whose size is $k - n$ bits. In principle, this architecture could remove any type of duplicate as long as the hash function take into account such scenario. What is more, HLS can be used for a part of the design, however the core of this work is a handcrafted design which uses many low level optimizations, consequently, we avoid using HLS because the tool is not thought for such low level optimizations, at least is not an easy task to achieve them.

5.4.1 Related Work

Authors in [117] introduced the packet duplication problem. They studied the different types of duplicates that can be generated when a mirror port is used to capture traffic. The main problem that packet duplication brings is an increase in the amount of information to handle, which makes more difficult the network monitoring process. We took this work as a starting point to create an FPGA architecture that deals with the problem of packet duplication at 100 Gbit/s networks. Packet duplication means that two or more packets contains the same information. However, this definition does not imply that some fields inside protocol headers could change —*e.g.*, such as IP addresses or ports. Anyway, in this work we focus on packets that look exactly the same, which is the common behavior for switching duplicates.

Comparing each byte of the payload, as it is mentioned in [117], is a complex task, because it implies parsing each packet header in order to get a pointer to the first byte of the payload. In the literature there are extensive research about packet parsing using FPGAs [16, 19, 118, 119], this extra complexity has been avoided for this proof-of-concept. Nevertheless, including packet parsing is an appealing alternative for future work. On the other hand, the previous N packets should be stored to perform the comparison, and to guarantee line rate the packets should be stored on on-chip memory, a very scarce resource on FPGA. In doing so, the maximum number of packets is limited. What is more, the payload comparison depends on the current payload size and the size of stored packets, consequently, the comparison time is not deterministic. Hence, to address the issue of storing the whole packet with variable size, we decided to perform a hash

function to every packet and obtain a fix-length digest.

Noteworthy, this problem is abbreviated to find efficiently an element within a memory structure of S elements, and that also allows a very fast insertion and deletion of elements. Currently, there are different technological choices to satisfy these requirements, following the most popular options are summarized. Content-Addressable Memory (CAM) [120] is a memory that compares input data against a set of stored values, and returns the address of the matching data, if the data is present in the memory. Usually, CAMs feature a single-cycle latency lookup time, thus making them a really fast option, although the speed of a CAM comes at the cost of a huge resource utilization. Authors in [121] introduce the architecture of CAMs and their features. As a generalization of CAMs, Ternary Content-Addressable Memory (TCAM) [122, 123] is one of the most popular methods for packet classification. Using the concept of *"don't care"* in the comparison, it allows searching for not exact matches. This sort of memory is widely used in network equipment to route traffic, because it is very useful to compare subnetworks. As well as CAMs look up time is one clock cycle but resource utilization is significant. Additionally, removing stored elements in a TCAM is a difficult task, as it takes longer than comparison [122].

Furthermore, another popular alternative is Bloom Filters [124], this data structure is widely used to test if an element is within a set. Due to its functionality, false positive matches are possible, on the other hand, false negative matches are not. While it is straightforward to insert elements, removing a single element is not possible for the simple reason that it could remove other elements as well. The latter reason discards such data structure for the problem at hand.

Finally, there is a commercial solution in the market [125], which claims to be capable to detect all duplicate packets and remove them at 100 Gbit/s speed. However, they do not provide any details on how they achieve such functionality, or what is the size of the sliding window in terms of time or number of elements.

The CAM appears to be the best option. However, to fulfill the deduplication requirements, an element has to be removed from the table after a certain amount of time or number of incoming packets. Moreover, a second memory is mandatory to implement the removal logic (a sorted list with the elements within the CAM), this increase the amount of required memory and due to the stringent restriction of this problem that memory has to be on-chip. Consequently, using CAM seems not to be the ideal solution. Accordingly, we propose a novel architecture able to tackle packet deduplication taking advantage of BRAM memory in the most efficient possible way. Another caveat of CAMs is that the insertion time depends heavily on the load factor.

In addition, authors in [117] recommended to use a time-sliding window of 15 ms to detect packet duplication in 100 Mbit/s networks. According to that work, it is expected

that such time window is going to be several times shorter at 100 Gbit/s, due to faster switching times —for instance, some switch vendors claim that their equipment can switch frames in the order of microsecond. Nevertheless, we decided instead to use an element-based sliding window, because for time management we would need extra bits to store timestamps and a mechanism to remove expired packets, thus increasing both the complexity and on-chip memory requirements of the design.

5.4.2 Hashing a Packet

Storing and comparing variable-length packets is neither efficient nor deterministic, because the Ethernet frame length could range between 60-Byte and 1518-Byte (not counting the Frame Check Sequence). In order to solve this problem, hash functions are commonly used to reduce the contents of a packet to a fixed-length key (digest). There are different sorts of hash functions, some better than other [126]. In the context of networking, a very important metric of hash functions is how the packets are distributed along the addressable area of the hash function [127]. A hash function is judged as good if it produces a uniform distribution of packets into the addressable area. However, achieving this goal can sometimes be complex because each network link has its particular traffic patterns.

Hash functions suffer from a problem called hash collision, that is, that two different inputs to the hash function produce exactly the same digest. In our case, this means that several packets can have exactly the same key. Software implementations typically deal with this problem using either chained hash tables or a fixed array of M possible collisions. In both cases, it is necessary to store the original data along with the hash in order to solve the collision. The difference is that the former uses dynamic structures, while the latter uses static ones. In principle, the latter is more suitable for FPGA implementation, but some type of insertion and replace policy has to be implemented when no more space is available in the fixed array of M possible collisions.

Hash collisions are particularly detrimental to our solution of packet deduplication. For the problem at hand, two equal digests mean a duplicate packet. However, in case of collision, two different packets will have the same digest, consequently, generating a false positive. This is an undesirable condition because it will remove a non-duplicated packet, thus potentially losing valuable information. In order to minimize the probability of a false positive occurring, we analyze the relationship between the digest size and number of packets in the element-based sliding window. Equation 5.1 is a fair approximation of the probability that two different elements in a window of S elements have the same key in an addressable area of size K. A detailed background about the equation can be found in [128].

$$P_{collision} = \frac{S^2}{2K} = \frac{S^2}{2x2^{key-bit}} \tag{5.1}$$

Given the equation above, we face a conflict in the number of elements in the window. The more elements we have, the bigger the window is, therefore, bigger delay can exist between duplicates. However, the probability of collision rises quadratically with the number of elements, so there exists a trade-off between window size and probability of a false positive. In our case, we chose as a starting point a 64-bit key ($K = 2^{64}$) and a 65,536-element window ($S = 65,536$), which gives a probability of collision of approximately $1.164 \cdot 10^{-10}$, which is low enough for our problem.

In the bibliography [129–131] there are many suitable implementations of different hash functions for FPGA designs, and also more complex schemes, such as sketch tables [132], but none of them operates at 100 Gbit/s — at least in a pipeline fashion with an II=1. Thus, for this proof-of-concept we decided to use a simple hash function, however, this piece can be upgraded at any time. What is more, this piece of hardware can be implemented using HLS in particular for complex hash functions.

5.4.3 Architecture Overview

The trivial FPGA implementation of an element-based sliding window consists of using a shift registers with M stages, and comparing each stored digest against the current digest in parallel. This solution could be easily implemented using LUTRAMs, but it is only valid for a few elements. If the number of stages of the shift register grows, the resource utilization grows too, thus adding excessive complexity to the FPGA design. As a result, the design frequency has to be decreased in order to meet time constrains, and a 100 Gbit/s data rate cannot not be achieved. Actually, it is absolutely impossible in current FPGAs to compare 65,536 elements concurrently in one clock cycle — as small as 3.1 ns.

We developed instead a mixed alternative. On the one hand, it takes advantage of BRAMs and their true dual port capabilities. On the other hand, we mimic a shift register, to emulate the behavior of an element-based sliding window. The main idea is to connect the data output of $stage(i)$ to the data input of $stage(i+1)$, so that when the write enable signal is asserted, data from the i-th stage passes to the $(i+1)$-th stage and so on, producing a shifting of the elements of a given row, and also dropping the oldest element. Now, instead of only one shift register, we have N shift registers selectable by the address of memories. We use the n most significant bits of the key to address the memory and the remaining bits of the key are stored in the memory. Therefore, N is always a power of two. What is more, part of the comparison is done when addressing

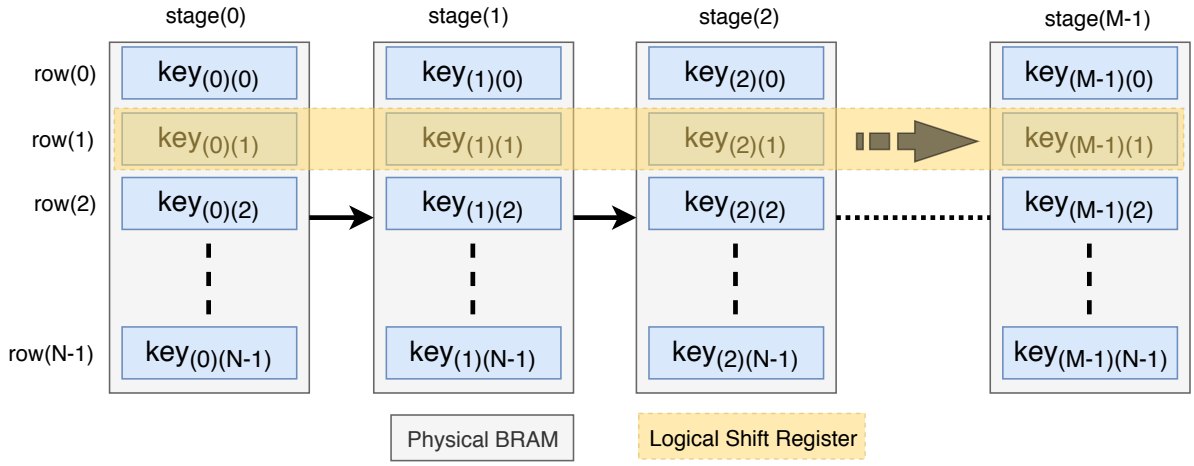


Figure 5.6: Architecture of the memory with N rows and M stages.

the memory with the most significant bits of the key.

Figure 5.6 shows a high-level overview of this architecture. First, each row behaves as a different shift register, thus increasing the number of the elements in the window. Second, it allows sharing the comparison logic to detect duplicates among all rows (that is, all shift registers), because only one row will be active at a given time. However, it should be noted that this scheme does not guarantee an even access to all rows. Depending on the traffic patterns, the hash function might tend to access some rows more frequently than the others. Thus, we cannot state that 100 % of the elements are going to be always used. Anyway, this solution offers a larger capacity than a simple shift register.

We decided not to use HLS to implement this architecture, mainly due to the low level optimization needed to achieve an initiation interval of one. Designing such low level optimization is easier with traditional HDL.

5.4.4 FPGA Architecture

Figure 5.7 shows a block design of the deduplicate module and the connection between its submodules. As the hash function is computed for the whole packet contents, the design follows a store and forward approach.

The **Hash Function** module computes a digest of the packet using a hash function. It is straightforward task because we have chosen a naïve hash function: it simply divides the packet contents in 64-bit chunks, and it does a bit-by-bit XOR of all the chunks in order to obtain a 64-bit key. Though this hash is not as good as others are, it is enough for proof-of-concept purposes. However, any other hash function could be implemented as long as it has a pipeline implementation, or, if the initiation interval is different from one multiple implementations can be used. The resulting 64-bit key is outputted one

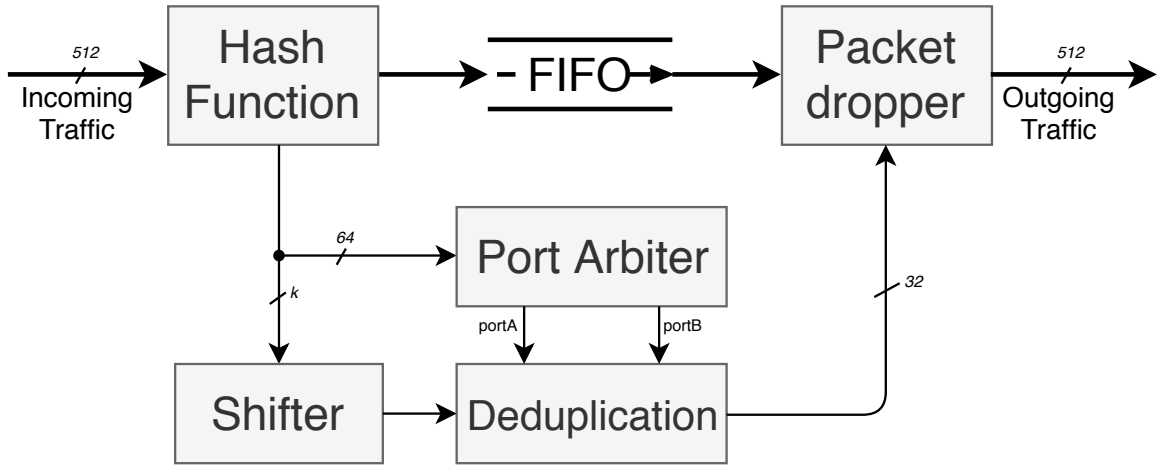


Figure 5.7: Global architecture of the deduplicate module.

clock cycle after *tlast* is asserted in the incoming AXI4-Stream interface. Meanwhile, while the hash function is computed, the packet is being queued in the FIFO, waiting for the drop or forward decision depending whether it is a duplicate or not. We implement this piece using HLS.

The **Packet Dropper** is a simple module, which waits for the decision coming from the Deduplicate module. When the decision is made, it starts to read the packet stored in the FIFO. If the decision was to drop the packet, the packet will be dequeued, but not sent. Note that it is mandatory to dequeue the packet in order to empty the FIFO. On the contrary, if the decision was to forward the packet, the entire packet will be dequeued and sent to outgoing traffic. We used HLS for this part of the design.

The **Port Arbiter** module selects what port of the BRAMs is going to be used. Usually, it works as a Round Robin mechanism, that intersperses each key to one of the two ports of the BRAMs. The reason for using the two ports is doubling the performance. However, if two consecutive keys point to the same row (the n most significant bits are equal), the both keys will be sequentially sent to the same port, to prevent data corruption. If both ports use the same address simultaneously, data integrity cannot be guaranteed.

The **Deduplicate** module is composed of M BRAMs (stages). The n most significant bits of the key are used to address the memories, while the remaining $k-n$ bits are compared with the data output of the M memories. If there is a match, the current key will not be written, and there are two possible actions regarding the element that matches: **A)** do nothing and keep that element in the memories; or **B)** remove it and create a bubble (gap) in the memories. Additionally, this module informs that the current packet is a duplicate and needs to be dropped. Later we will explain the implication of choosing **A)** or **B)** options. On the contrary, if there is no match, the $k-n$ bits of the current key are written to the BRAM at stage 0, and data from BRAM at stage i is

written to BRAM at stage $i+1$.

Finally, the **Shifter** module is a low priority process that is in charge of generating dummy elements to force a shifting of elements in the sliding window. This process is designed to remove old packets from the BRAM-based shift register. It uses a BRAM memory where each position contains the timestamp of the last access to each row. If we assume that the hash produces a uniform distribution, the mean access time to each row will be the maximum time of a packet × the number of rows ($pkttime_{max} \cdot N$ = $1230.4ns \cdot N$). The checker process inserts a bubble when a row has not been accessed for this time. Therefore, this module helps to reduce the false positive ratio by removing the oldest key.

We used HDL to implement the **Port Arbiter**, the **Deduplicate** and the **Shifter** modules because we need low level optimizations and determinism, which cannot be guaranteed using HLS.

5.4.4.1 Low Level Architecture of Deduplicate Module

Figure 5.8 shows in more detail the low level architecture of the BRAM-based shift register. As it was explained above, the current key is latched and separated in two parts, where the n most significant bits are used as address to select one of the N shift registers. Additional registers have been placed at the output of BRAMs due to the relatively high clock-to-output time of these memories ($T_{RCKO\_DO} \approx 1.35ns$ in the Xilinx UltraScale Family). Without these registers it would be very difficult to meet timing, but the drawback if that read latency is increased to 2 clock cycles. Once that the output data is at the registers (two clock cycles after setting the address), its values are compared with the remaining $k-n$ bits of the current key. If there is a match in any of the comparisons, it means that a duplicate has been found. That is, as a result of the parallel comparison, we have a vector of M bits pointing (one-hot encoding) at the position of the element that matches (if it was any). This vector is registered, and in the next step a bitwise OR reduction is made. This OR reduction is critical in terms of timing, and its results is the duplicate found flag. If this flag is true, the current key will not be inserted in the first element of the shift register —$wea_0 = 0$.

As stated above, there are two options when a duplicate is detected. **A)** Do nothing with the element that is stored in the memory: In this case the blue multiplexer in Figure 5.8 is not implemented. **B)** Overwriting the position where the match was found, creating a bubble. The comparison bit of the i-th stage is connected to the select pin of the i-th multiplexer. When a match happens, the element in the i-th stage will be deleted. This feature is a good idea when we are sure that there is only one duplicate, as it creates space for other elements.

When there is no match, there are two possibilities as well. If we have decided to use alternative A, the write enable signals for all M BRAMs are asserted, thus producing a shift of the elements from the i-th stage to the $(i+1)$-th stage, and also eliminating the oldest element. However, if we chose alternative B, the row could include bubbles (each element has a valid flag). If we assert the all write enable signals, a shift will be produced removing the oldest element, but this has no sense, because we are wasting space. Instead of asserting all write enable signals, we propose using a smarter logic —*e.g.*, a priority encoder. When a new key arrives and there is not match, we use the valid flag to calculate the value of the write enable vector. For example, if the j-th position is empty, the priority encoder asserts the write enable signal only for stages 0 up to j-th. This pokes the first bubble and does not remove the oldest element. Although this solution is straightforward, it however requires for each stage an OR reduction of the valid flags of the previous stages. That is, that for the worst case ($Stage_{M-1}$) we need an OR reduction of $M-1$ bits, which might be challenging considering the tight clock period.

Finally, the shift bit connects to the Shifter module and it is used to insert bubbles in case that the incoming traffic pattern is such that certain rows are not updated. It takes only one clock cycle to insert a bubble in a given row. All write enable signals are asserted, and the select pin of the multiplexer for $Stage_0$ is set to "1" so that a bubble is inserted in the first stage, while the select pin of the multiplexers of the remaining stages are set to "0" in order to shift the elements.

As a summary, the deduplication process takes 4 clock cycles. Anyway, if two consecutive addresses are the same, then we can save one clock cycle at the fetching process, because the address has already been set. Due to this latency, we decided to use both ports of the BRAMs in order to get the maximum throughput, though this decision implied additional logic in order to avoid conflicts, as it has been explained in the previous section.

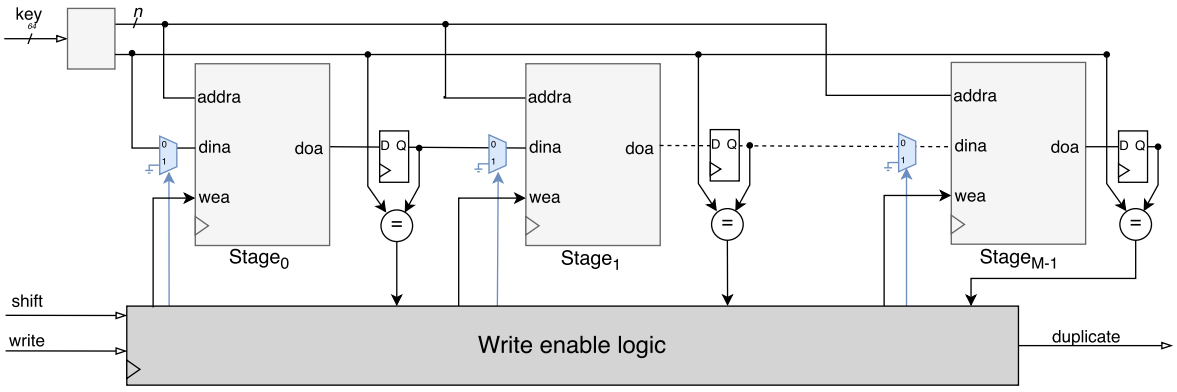


Figure 5.8: Deduplicate architecture based on M stages.

5.4.5 Experimental Results

First of all, we targeted the VCU108 board (section 3.8.1) for this proof-of-concept. Apart from that, we completed several implementations of our architecture, varying the $Depth$ and $Stages$ parameters. Table 5.3 presents the results for five different combinations of Depth and Stages parameters for which timing constraints could be met. We have highlighted our initial goal of 65,536 elements, but a larger number of elements is also possible if more stages are used. The selected configuration of BRAMs is 10-bit address and 72-bit data, so memories are underutilized if the Depth parameter is less than 1024. Additionally, 17-bit out of the total 72-bit of data are not used; these bits could be used to store an extra hash in order to reduce the false positive probability. FPGA resources utilization for all combinations are under 10 % except for BRAMs. Therefore, there are enough resources to include additional features, such as a traffic filters (section 5.3 [133]).

5.4.6 Packet Deduplicate Pipeline Architecture

The previous low level architecture relies in the fact that, always there are two clock cycles between packets. This is true for 100 Gbit/s link speeds, where the minimum time between packets can be as small as 6.72 ns. However, for higher network link speeds this will not be true anymore. Consequently, we have improved the low level architecture to handle one hash comparison per clock cycle — initiation interval equal to one. To do so, we use one of the ports of the BRAM for lookup and the other for insertions. As explained before, to reach high performance with BRAM the output register is mandatory, therefore, the read latency is two clock cycles, which guarantees higher performance. What is more, the insertion latency is one clock cycle. Hence, this new implementation in the worst case scenario has an initiation interval of three clock cycles, when two consecutive digests target the same row. Therefore, to achieve II=1, we have included a four-stage shift register in parallel to the BRAM-based to overcome the write and

BRAM-based Shift register			packet size (bytes)			Used BRAMs
			60	760	1,514	
Depth	Stages	Elements	Sliding Window (ms)			
256	64	16,384	0.11	1.02	2.01	128
512	64	32,768	0.22	2.05	4.03	128
1,024	64	65,535	0.44	4.11	8.06	128
1,024	75	76,800	0.51	4.81	9.44	150
1,024	78	79,872	0.53	5	9.82	156

Table 5.3: Summary of the schemes of $Depth$ and $Stages$ that meet timing and its maximum sliding window as a function of the packet size.

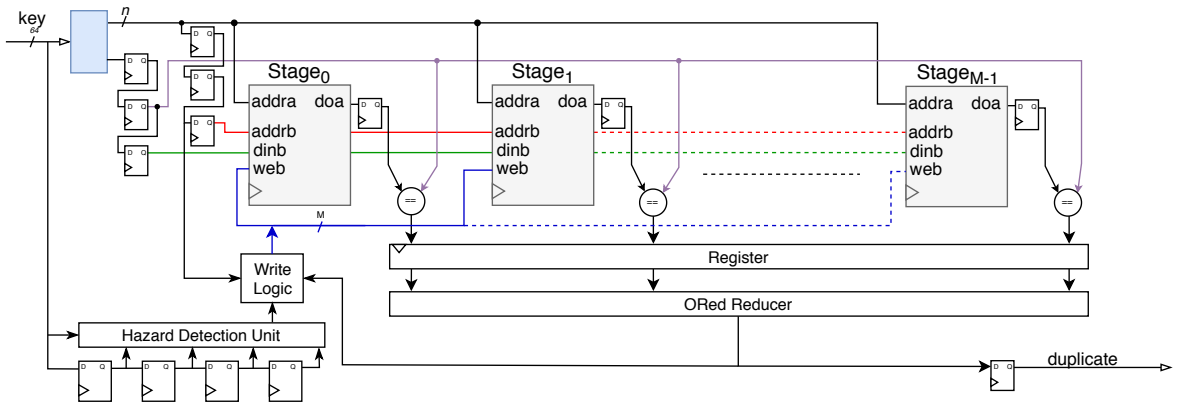


Figure 5.9: BRAM-based shift register, packet deduplicate pipeline architecture with hazard detection unit.

read latency of the BRAMs. Note that there is another register stage after the digest comparison (ORed Reducer). Consequently, the comparison is not only done against the output of the BRAM-based shift register, but also, against the auxiliary four-stage shift register. In terms of processors, this auxiliary piece of hardware handles the data hazards. The other port of the BRAM is used to insert new digest when such digest was not found within the memory. As a result, this new architecture is able to handle a maximum of or 322 million packet per second. Figure 5.9 depicts this improved architecture. Noteworthy, this architect is highly pipeline therefore we can increase the number of stages, number of BRAM in the memory.

We implement this pipeline architecture targeting the VCU118 board (section 3.8.1). We used Vivado 2018.3 to synthesize and implement the design. We were able to meet timing with 150 stages and depth of 2,048. In this case, the resource usage of the pipeline architecture is as follows: 0.37 % LUTs; 0.25% FF and 28.19% BRAM. Therefore, we can store up to 307,200 elements in our memory — collision probability of 2.55×10^{-9}— that means a time window of:

- 2.06 ms with 60-Byte packets, worst case scenario.
- 19.26 ms with 760-Byte packets, mean packet size.
- 37.79 ms with 1514-Byte packets, largest packet (without taking into account jumbo frames).

5.5 Conclusion and Discussion

In this chapter we have presented two bump-in-the-wire passive solutions to thin network traffic. On the one hand, a stateless architecture to identify and filter encrypted

traffic at 100 Gbit/s was presented. The algorithm is based on the recognition of sequences of consecutive characters or the presence of a percentage of readable characters in the payload of a packet. This hardware solution is built on top of a previous work. Compared to the original work, there is a tenfold performance increase while evaluating the whole packet. The current architecture can be easily extended to discriminate any kind of traffic by simply modifying the decider module (Figure 5.2). Additionally, it would be possible to add more stateless filters (such as IP/port origin/destination, protocol, etc.) in combination to this filter, while, keeping the line rate operation. Although this architecture is aimed for network monitoring purposes (capturing and storing unencrypted traffic), it can also be used to divert binary traffic to network nodes with decryption capabilities for further deep packet inspection. Based on the literature 70% of the traffic is encrypted and the mean packet size in a network is around 700-Byte, thus in a full link this implementation reduces the output mean packet size to 255-Byte. A 65% reduction on the storage capacity.

On the other hand, we have presented a novel FPGA-based architecture capable of detecting and removing duplicated packets from a mirror port in Ethernet network traffic monitoring system. Deduplication is important, because it means less CPU load and less storage requirements in the network monitoring equipment. The architecture uses packet hashing and an element-based sliding window. In the basic architecture, the size of the sliding window corresponds to of 0.53 ms with minimum-size packets, or 9.82 ms with maximum-size packets. However, the improved architecture is able to store 3.84× more keys. According to the bibliography, such times are big enough to successfully remove switching-based duplicates. We presented a novel scheme of memory that we have named BRAM-based shift register. This memory scheme allows us to keep N shift registers of M stages with M elements simultaneously available. Additionally, we used the two ports of BRAMs in order to access two different shift registers in parallel and thus double performance. In devices with multiple Super Logic Region (SLR) we could use a chain of deduplicate modules to increase the number of stored elements without jeopardizing the performance of the architecture.

Both architectures (section 5.3.2 and section 5.4.4.1) are designed to work at line rate in 100 Gbit/s Ethernet links (up to 148.8 million packets per second), using a 512-bit AXI4-Stream interface clocked at 322.265625 MHz. Such data rate calls for a careful design of the critical parts of the algorithm. A mix design methodology that uses HLS for non-critical parts and HDL for time-critical regions is able to cope with that data rate, but at the same time it provides a significantly better productivity than a conventional, HDL-only approach. What is more, we also observed that for handcrafted architectures as well as for well-known low level architectures HDL is still the best alternative to design.

Moreover, these two implementations can be chained together in the same FPGA to further reduce the traffic load in the software probe.

Part III

Offload Tasks

CHAPTER 6

CHECKSUM OFFLOADING

End-to-end packet integrity in TCP/IP is ensured through checksums based on one's complement addition. Once a negligible part of the overall cost of processing a packet, increasing network speeds have turned checksum computation into a bottleneck. Actually, supporting 100 Gbit/s bandwidth is a challenge partially due to the difficulties of performing checksums at line rate. As part of a larger effort to implement a 100 Gbit/s TCP/IP stack on an FPGA, in this chapter we analyze the problem of checksum computation for 100+ Gbit/s TCP/IP links and describe an efficient solution for the 512-bit wide, 322 MHz buses being used in the 100 Gbit/s Ethernet interfaces of Xilinx UltraScale(+) devices. The proposed architecture computes thirty-three 16-bit one's complement additions in only 3.1 ns, more than enough to support 100 Gbit/s Ethernet links.

6.1 Introduction

The TCP/IP network protocol provides mechanisms to verify data integrity through the detection of corrupted packets. These mechanisms are based on computing a checksum of the packet at the source, before sending it, and then controlling the checksum upon arrival at the destination. Checksums are used to protect both the packet headers as well as the payload of the packet, and there are both layer 3 (IP headers) and layer 4 (TCP segments or UDP datagrams) checksums — the layer levels refer to Open Systems Interconnection (OSI) model (Figure 7.1). In both cases, the TCP/IP checksum is

calculated as 16-bit one's complement addition over 16-bit words.

At low line rates, the overhead of computing the checksum is expensive but not necessarily overly large when compared to that of other parts of the protocol. Yet, already early in 1989 Clark *et al.* [134] suggested that the checksum could become a problem in packet processing. As network bandwidth increases and, thus, the amount of data being sent and received per unit of time grows, in-flight computing of the checksum becomes a major bottleneck and might affect the overall latency. In particular, the checksum must be computed not only at the source and destination, but also, during routing if there is any change to the header. In the case of TCP and (optionally) UDP packets, the payload is also included in the checksum. Hence, computing the checksum of a packet potentially involves processing many 16-bit words. As checksum calculation is performed several times over the life time of a packet, doing it efficiently is key to achieve higher line rates and lower latency.

Nowadays, at 10 to 100 Gbit/s speeds, packet processing is offloaded to hardware — usually to the Network Interface Card (NIC) — since otherwise processing is too slow. We carried out a simple experiment to confirm such hypothesis. We set up a testbed with a 128 GByte, 2.20 GHz Xeon E5-2630 v4 server (A) and a 192 GByte, 2.60 GHz Xeon 6126 server (B), both running Gentoo Linux (kernel 4.14.7). We used a Mellanox 100 GbE ConnectX-5 NICs on both servers, which were connected to each other via a QSFP28 direct attach cable, and we run `iperf2` over TCP to measure the overall performance. Results show that, when server B acts as a client, performance decreases from 28.3 Gbit/s to 10.0 Gbit/s when checksum offloading is disabled (*rx off* and *tx off* options of `ethtool`).

As part of a larger effort to implement a 100 Gbit/s TCP/IP stack on programmable logic (chapter 7), intended for in-network data processing and network-attached FPGAs. We have confronted the need of implementing an ultra-low latency checksum mechanism capable of sustaining a 100 Gbit/s line rate. While, the checksum arithmetic is well-known and comparatively simple, quoting the RFC793 [135] *"The checksum field is the 16-bit one's complement of the one's complement sum of all 16-bit words in the header and text"*, the low-latency requirement imposed by higher bandwidth is a major challenge. In this chapter we describe our solution to compute the checksum at 100 Gbit/s on an FPGA. We aim for an implementation with one clock cycle latency and one clock cycle initiation interval in order to support maximum throughput even for the shortest packets. Our design covers computing the checksum for i) the IP header: from 10 to 30 × 16-bit words, to be processed in one clock cycle; and for ii) the UDP/TCP/ICMP header plus payload: iterating over 33 × 16-bit words checksum per clock cycle over several clock cycles. The target frequency is 322 MHz (3.1 *ns*), which is the one used by the Xilinx's Integrated CMAC. Such an architecture is not suitable for HLS, due to the low level nature and necessary optimizations. The results are available as two open source implementations,

one for the IP header and another for ICMP, UDP, and TCP packets.

6.2 Related Work

The checksum computation is a straightforward algorithm, described in RFC793 [135] and RFC1071 [136]. Nevertheless, the overhead of checksum processing when implemented in software is well-known [137, 138]. Kay *et al.* [139] show that checksum calculation was the major processing overhead in a software TCP/IP implementation. Today's commercial NICs offer TCP/IP offloading, reducing the overhead on the CPU for any task related to packet processing, including checksum. Indeed, the Xilinx AXI 1G/2.5G Ethernet subsystem also provides full checksum offloading capabilities. Unfortunately, these capabilities are neither available in the 10 Gigabit Ethernet Subsystem nor in the UltraScale(+) Devices Integrated 100G Ethernet core.

One's complement addition has the interesting property of being associative. Thus, the 16-bit words can be added in any order. Although a hardware implementation was already suggested in 1996 RFC1936 [140], there have been only a few studies of low-latency checksum in hardware implementations, mostly for 10+ Gbit/s link speeds. Henriksson *et al.* [141] provide a 0.18 μm ASIC implementation for 10 Gbit/s Ethernet. On FPGAs, a Stratix III has been used to achieve a throughput of 14.2 Gbit/s [142]. Atomic Rules [143], along with other companies, offers a 10 to 400 GbE UDP Offload Engine with an integrated checksum computation for the UltraScale(+) families, but no details about the implementation are available and only the 10 Gibt/s and 40 Gibt/s versions seem to be accessible at the moment. Recent implementations of a 10 Gbit/s TCP/IP stack for FPGAs include checksum computation [30, 144] using HLS, however at such speed, there are at least 67.2 ns to compute the checksum an order of magnitude bigger that at 100 Gbit/s. The lack of alternatives for such a speed, and aiming for an initiation interval of one as well as one clock cycle of latency, lead us to study and implement an efficient Register Transfer Level (RTL) circuit to tackle the issue at hand. Therefore, in this chapter we exploit the Xilinx's Ultrascale(+) architecture to perform efficiently the TCP/IP checksum computation at a minimum line rate of 100 Gbit/s.

6.3 Use Cases

The results of this chapter are part of a larger effort to implement an open source TCP/IP stack capable of reach 100 Gbit/s line rate (presented in chapter 7). Other projects that can benefit from the contributions of this work are, for instance, a 10 Gbit/s TCP/IP stack [144], which we use as starting point, or projects using a NetFPGA [145]. In terms

of networking protocols, we have focused on IPv4 (Internet Protocol Version 4), since the deployment of IPv6 is still limited [146], and actually IPv4 is a more difficult problem because the IPv6 header does not include a checksum.

We make no assumptions about the use of the stack, regarding whether it is on a NIC, as an end point of the network, or on a router or on a middle box, because of that we aim for one clock cycle of latency. In such cases, checksum computation is used quite often. For instance, routers recompute the checksum if headers change as a result of a Network Address Translation (NAT) or port re-assignment. In NICs with offloading support, the host sends data through PCIe and the NIC has to segment and packetize such data, performing the checksum computation and including the result in the packet header. Finally, in accelerators using TCP/IP as a mean of communication, *e.g.* [30], outgoing packets must include both IP and TCP checksums and the checksum in the incoming packet must be verified before being processed.

An efficient checksum architecture boosts the performance of those implementations, reducing the latency and increasing the throughput, a win-win situation.

6.4 Module Communication and Processing Time

From now on, we will assume that the checksum implementation is interfaced with a 512-bit AXI4-Stream and its output is a 16-bit AXI4-Stream. In case of messages that span more than 64-Byte (*e.g.*, the checksum of a TCP segment), the message is split in as many 512-bit AXI4-Stream transactions as necessary, the tlast signal states when the message ends.

In TCP/IP over Ethernet, the shortest and largest packet are as follows. For small packets, the size is 60-Byte and time between packets can be as small as 6.72 ns. In a packet with a Maximum Transmission Unit (MTU) of 1500-Byte, the minimum time between two packets is 121.92 ns.

6.5 IPv4 Header and TCP/UDP Checksum Calculation

Following the RFC793 and RFC1071 [135, 136] specifications, the transmitter side computes the checksum as follows:

i. The value of the checksum word (16-bit) is set to zero — since the checksum field is part of the packet for which the checksum has to be computed.

ii. The message is split into 16-bit words.

iii. All 16-bit word fragments are added using one's complement arithmetic.

- The overflow bits are fed into the addition.

iv. The sum is complemented (flip the bits) and becomes the overall checksum.

v. The checksum is inserted into the header and sent with the data.

On the other hand, the receiver uses the following complementary steps for error detection:

i. The message (including the checksum field) is split into 16-bit words.

ii. Every word is added using one's complement arithmetic.

iii. The result is the computed checksum. If the value is 0, the message is considered to be correct.

The procedure is similar for the IP header, TCP header and optionally for the UDP header. In what follows, the differences between the three cases are explained and one's complement arithmetic is discussed in detail.

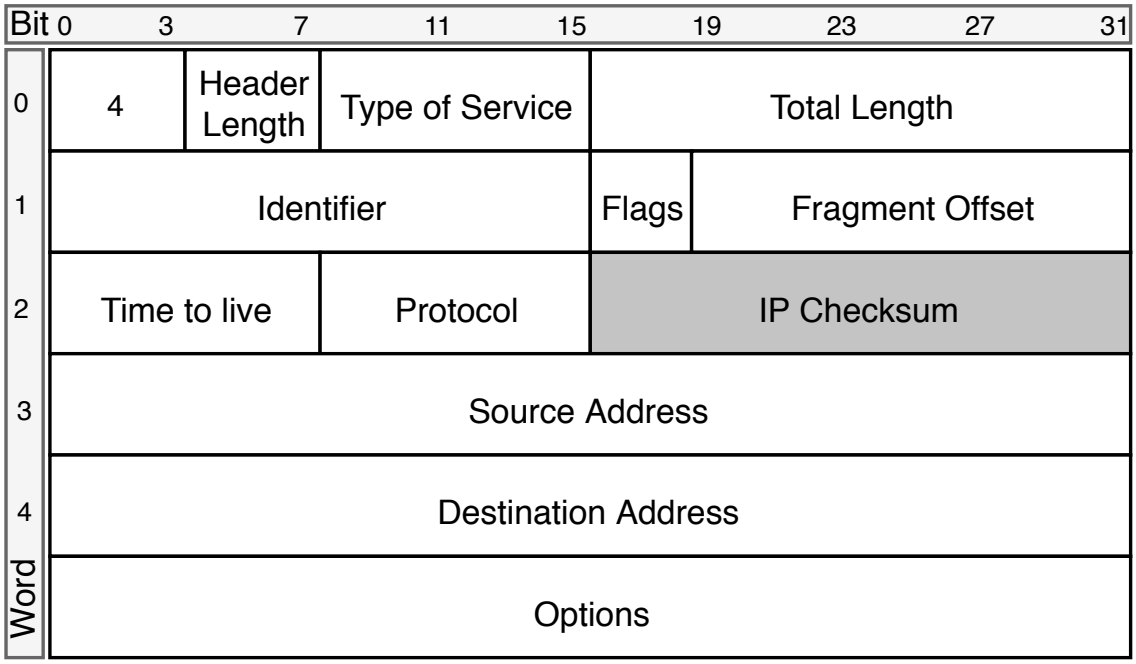


Figure 6.1: IP Version 4 header.

6.5.1 IP Header Checksum Computation

Figure 6.1 shows the IPv4 header with its 14 fields. Thirteen are mandatory, whereas, the 14^{th} field is optional and appropriately named: options. Since an IPv4 header may contain a variable number of options, the Internet Header Length (IHL) field (4-bit) defines the size of the header in steps of 32-bit words, which also coincides with the offset to the next protocol — OSI layer 4. The minimum value for this field is five which indicates a length of 5 × 32-bit = 160-bit ≡ 20-Byte ≡ 10 × 16-bit words. As a 4-bit field, the maximum value is fifteen words (15 × 32-bit, or 480-bit ≡ 60-Byte ≡ 30 × 16-bit words).

The interface of our module is 512-bit wide. As a result, the whole IP header fits in a single transaction. Processing complexity is mainly caused by the variable header length, which requires a variable sum ranging from 10 to 30 × 16-bit words. The proposed architecture considers the maximum possible number of words and a multiplexer selects whether the word is valid or not.

6.5.2 TCP/UDP Header Checksum Computation

TCP checksum is a 16-bit field in the header, Figure 6.2. On the contrary of IPv4 header, the checksum of OSI layer 4 covers a pseudo-header, the TCP/UDP header, and the payload. The pseudo-header has to be created prior to checksum calculation with information from the IPv4 header that includes: the source and destination IP

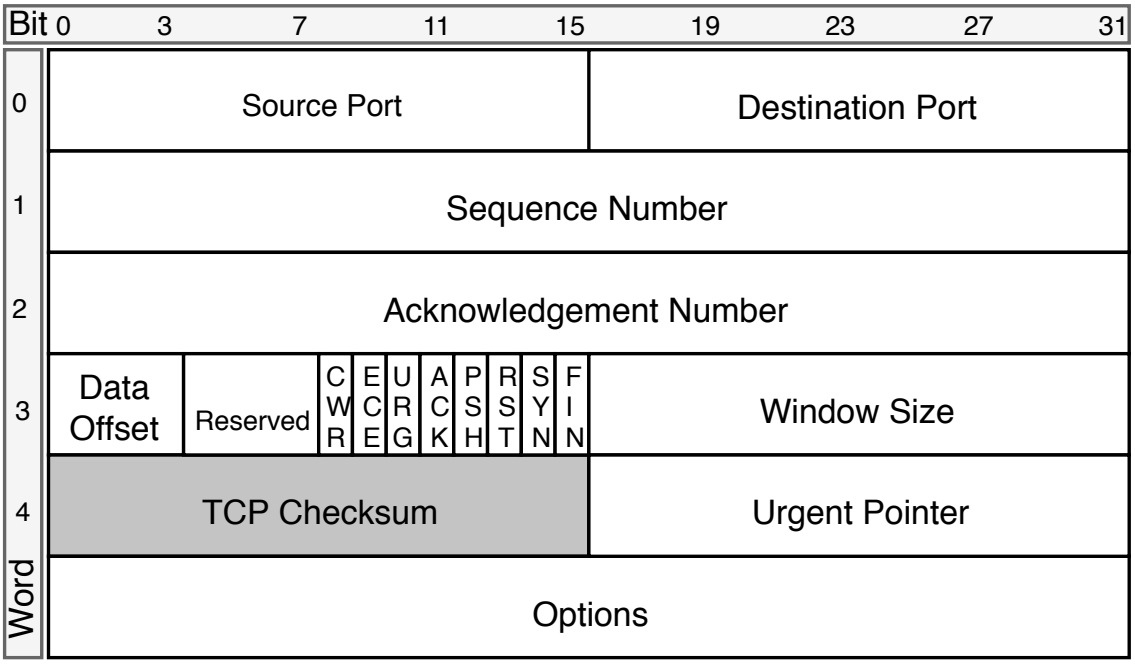


Figure 6.2: TCP header.

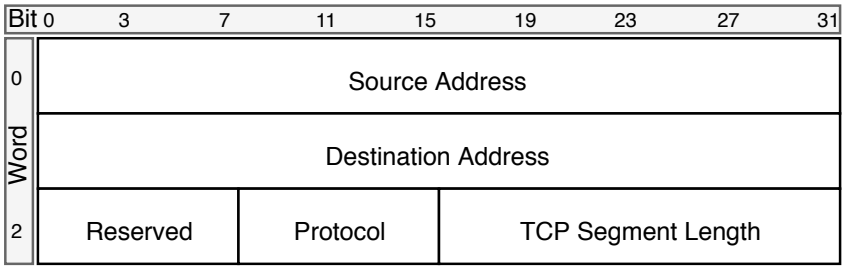


Figure 6.3: TCP and UDP pseudo header.

addresses, the protocol and the TCP segment length, Figure 6.3. The message covered by this checksum is shown in Figure 6.4. In case of odd number of bytes, a zero padding is inevitable to make the total number even. Note that the checksum field is at the very beginning of the header, meaning that the packet has to be stored and cannot be delivered until the last byte has been taken into account in the checksum computation. This is the main reason why the latency of the checksum computation impacts directly in the latency and performance of the network protocol.

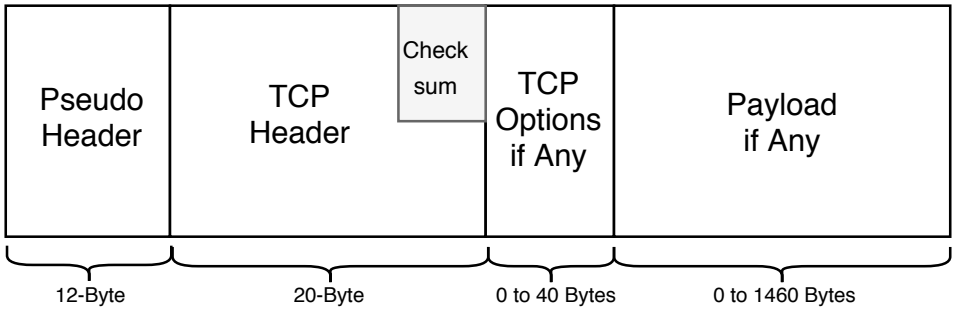


Figure 6.4: Data used to compute TCP checksum.

For UDP packets, the checksum is computed including the same pseudo-header as TCP, Figure 6.3. The UDP header size is fixed to 64-bit, which includes four fields, each of which is 2-Byte (16-bit), Figure 6.5. The maximum length of an UDP packet is 65,507-Byte (65,535 maximum IP packet size - 8-Byte UDP header - 20-Byte IP header). For the longest packet, 12 + 65,535-Byte = 65,547-Byte require 1025 × 512-bit AXI4-Stream transactions, we consider that case for a specialized network supporting jumbo frames. However, packets are hardly ever that long in a wide area network, because the Ethernet MTU is usually 1500-Byte.

In the case of TCP, the main difference with respect to UDP is that the TCP header might have optional fields, so its size varies between 20-Byte to 60-Byte in 4-Byte steps. In practical terms, the payload varies from 0 (acknowledge packets without payload) to 1460 (maximum segment size that fits the Ethernet MTU). Hence, the shortest packets have 12-Byte (pseudo header) + 20-Byte (header), *i.e.* can be processed in one 512-bit

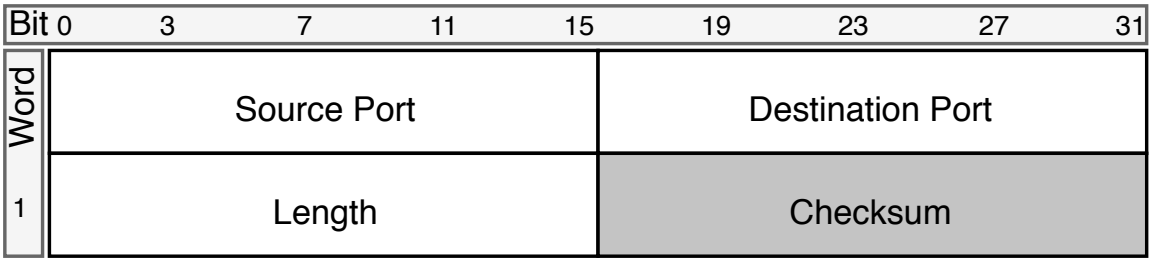


Figure 6.5: UDP header.

transaction. For the longest packets, 12 + 20 + 1460-Byte = 1492-Byte require 24 × 512-bit AXI4-Stream transactions.

6.5.3 One's Complement Addition

The one's complement of a binary number K in an N-bit representation system is determined by inverting every bit (flipping '0's for '1's and vice versa). It is arithmetically equivalent to perform $2^{N-1} - 1 - K$. Therefore, the number zero has two representations (00...00) and (11...11). As a consequence, an N-bit one's complement numeral system can only represent integers in the range $-(2^{N-1} - 1)$ to $2^{N-1} - 1$, whereas two's complement can represent integers in the range of -2^{N-1} to $2^{N-1} - 1$.

One's complement addition is calculated by summing as natural numbers and adding the carry to the result — *a.k.a.* swing the bit(s) around. An interesting property in multi-operand one's complement addition is the possibility to add as natural numbers (equivalent to two's complement) and "recirculate" (swing around) all the carries. As an example, consider the following 20-Byte IP header. The underlined 16-bit word represent the checksum.

```
4500  0030
0000  0000
4006  F96A
C0A8  0005
C0A8  0008
```

In order to calculate the checksum, we can sum each of 16-bit values within the header in a two's complement fashion, avoiding only the checksum field itself (considering as 0). Then we have the following addition (values in hexadecimal)

```
4500 + 0030 + 0000 + 0000 + 4006 + 0 +
C0A8 + 0005 + C0A8 + 0008 = 20693h (132755 dec).
```

Then swing the bits outside the 16-bit boundaries around getting 0693_{hex} + 2 = 0695_{hex}. The one's complement inverse (negation bitwise) is the checksum, *i.e.* not(695) = $F96A_{hex}$.

In order to verify the checksum, all 16-bit numbers are added including the checksum:

```
4500 + 0030 + 0000 + 0000 + 4006 + F96A +
C0A8 + 0005 + C0A8 + 0008 = 2FFFDh (196605 dec).
```

"Recirculating" the bits outside the 16-bit boundaries. $FFFD_{hex}$ + 2 = $FFFF_{hex}$. Taking the one's complement (flipping every bit) yields 0000_{hex}, which indicates that no error is detected.

Observe that is equivalent to add bigger numbers and reduce later to 16-bit. For instance, we can perform the same one's complement addition using 32-bit values instead — groping two 16-bit words regardless of the order —:

```
45000030 + 00000000 + 40060000 +
C0A80005 + C0A80008 = 20656003Dh
```

Then the number is reduced to 16-bit:

```
20656 + 003D = 20693h (132755 dec).
```

A final reduction is necessary 0693_{hex} + 2 = 0695_{hex}. The one's complement (not(0695) = $F96A_{hex}$) gives the checksum.

6.5.4 Solving the Worst Case at 100 Gbit/s

In case of TCP or UDP, several 512-bit wide transactions could be needed to process a packet. The processing involves adding 512/16 = 32 words of the current transaction, plus a 16-bit word from the previous computation, in case of a message larger than 64-Byte. Based on this, we propose a basic building block for the design consisting of an adder for 33 × 16-bit one's complement numbers in 3.1 *ns* (322 MHz). For longer packets, we combine this basic building block as needed. In what follows we study different alternatives to add these 33 numbers in one's complement arithmetic.

6.6 Architectures for Checksum Computation

In a first attempt, we modeled the problem using C/C++ in Vivado-HLS. Both a naïve and an advance version of the code needed at least several clock cycles to complete. Consequently, this approach was discarded. The next version of the solution was developed at RTL level, namely in VHDL.

6.6.1 Naïve Binary Computation

Xilinx FPGAs implement efficiently two's complement addition using built-in carry-logic resources. We name this approach *ArchBin16*. It uses a binary tree reduction, Figure 6.6, where six levels of adders are necessary to get a 22-bit number. Then two extra additions are needed to achieve the one's complement number.

6.6.2 Wide Binary Computation

With the purpose to reduce the critical path latency and leverage the low-latency built-in carry-logic, wider adders can be used in the first stages. This architecture is called *ArchBin32* and it is based on 32-bit words. It implements a binary tree reduction; four levels are necessary to obtain a 35-bit word. Then two extra levels are required to obtain the result. The total amount of additions is the same as before but the routing seems to be more complex.

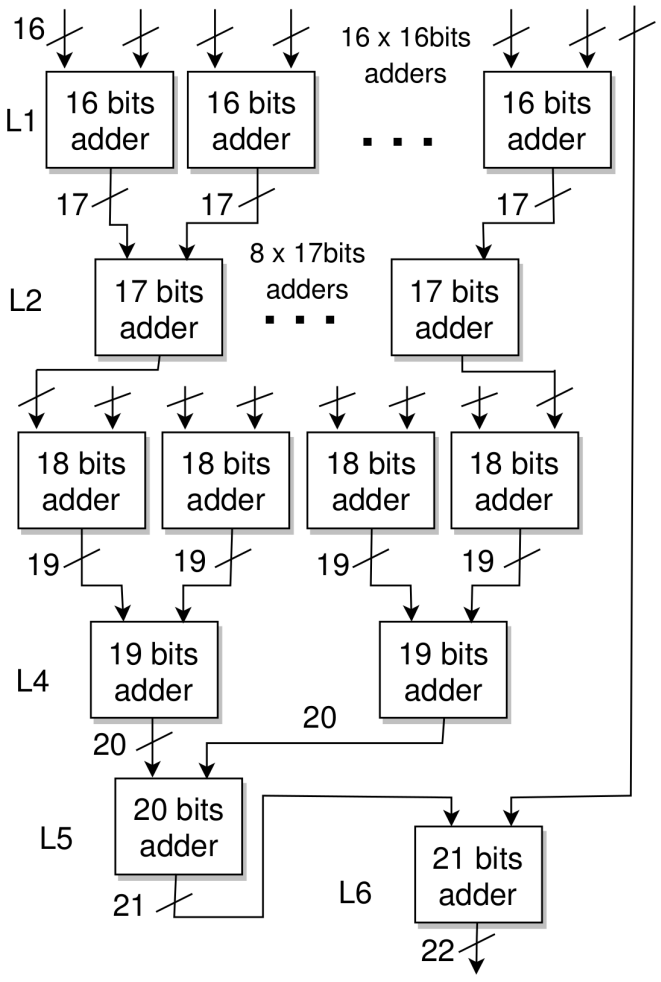


Figure 6.6: Binary tree adder.

6.6.3 Using Ternary Trees

Using the built-in carry-logic and the adjacent 6-LUT, it is feasible to build ternary adders (add 3 N-bit-2's complement elements) using the same resources as a binary adder. Taking advantage of this idea, it is possible to reduce the tree depth and the resource usage. The first Level (L1) reduces 33 × 16-bit numbers to 11 × 18-bit numbers. Thus, within four logic levels we get a 22-bit number. Then two supplementary additions are needed to achieve the one's complement number as stated previously. This architecture is called *ArchTern16*.

6.6.4 Using Reduction Trees

None of the previous alternatives reaches the desired performance. The solution that we have finally adopted leverages the Carry Save Adder (CSA) to reduce the numbers with minimum latency. In Xilinx FPGAs, it is possible to implement efficiently 7-3 counters (a CSA that reduces 7-bit to 3-bit number without carry propagation) [147]. Each bit of the result is implemented using two adjacent 6-LUTs and a slice multiplexer (muxF7), see Figure 6.7. In such figure a_6 to a_0 are inputs whereas c_2, c_1 and c_0 are the output of logic functions which combined represent the addition of the seven inputs. Harnessing such architecture, Figure 6.8 shows an example of a 7 to 3 CSA adapted for one's complement arithmetic. Seven n-bit words are arranged, the procedure groups the seven bits for each column (C_i), as result the sum is a 3-bit word per column (Re_i), which represents the amount of ones in the vertical column — sometimes this function is also referred as popcount. The weight of the least significant bit of the result is the same as the weight of the column C_i. Note that the two most significant columns could produce overflow bits, however, due to one's complement arithmetic properties the bits are swung around, filling the holes produced by Re_1 and Re_2. More in detail, the overflow bit of

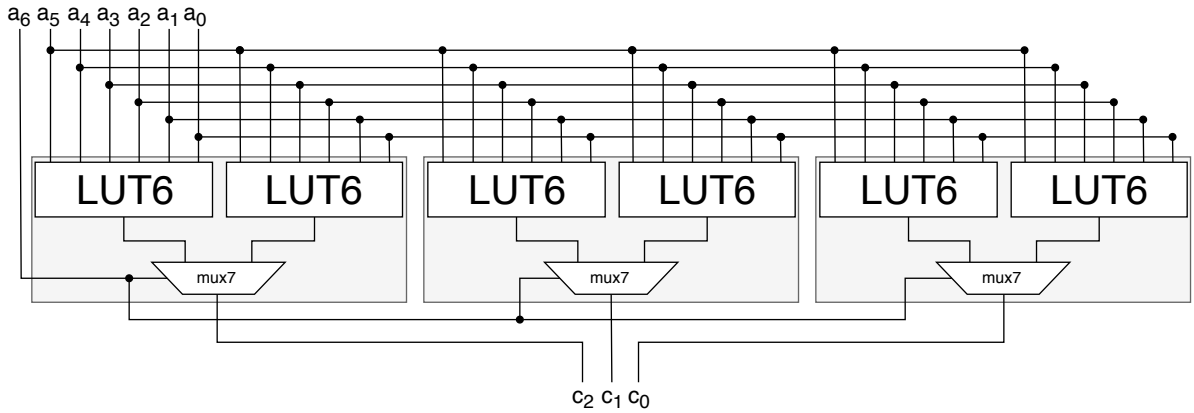


Figure 6.7: Carry save adder low level architecture using FPGA fabric logic.

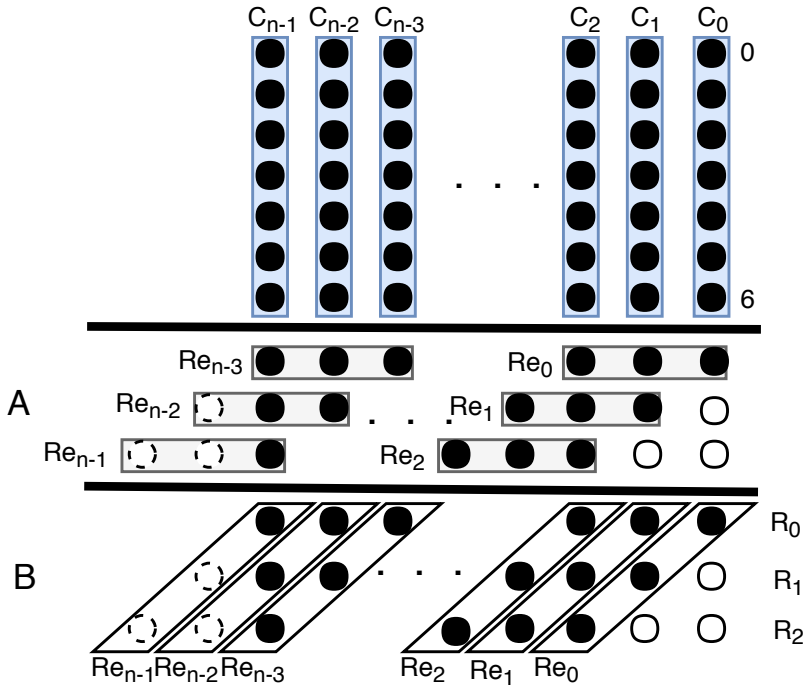
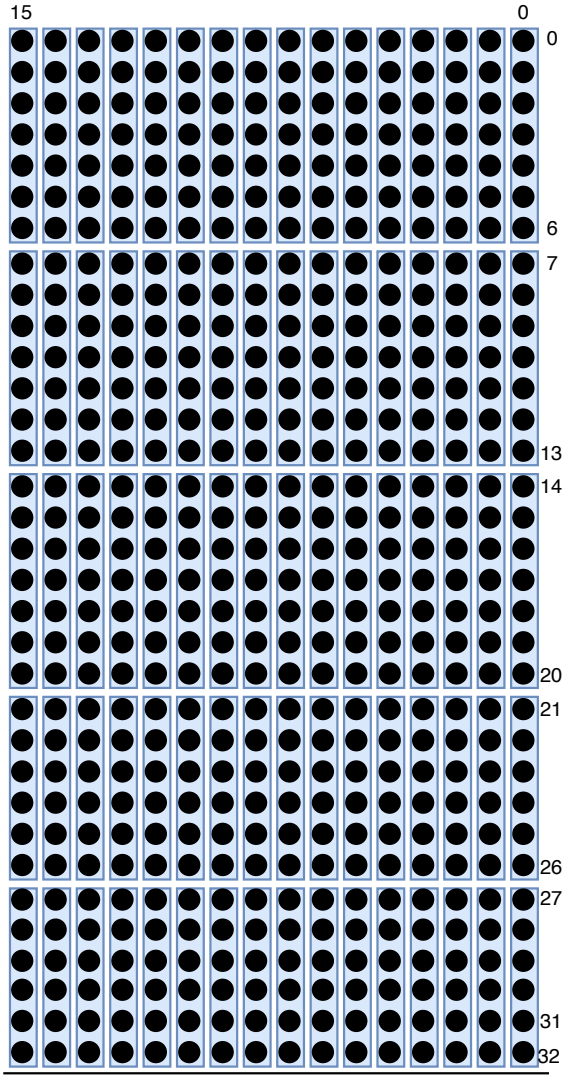


Figure 6.8: 7 to 3 carry save adder example adapted for one's complement addition.

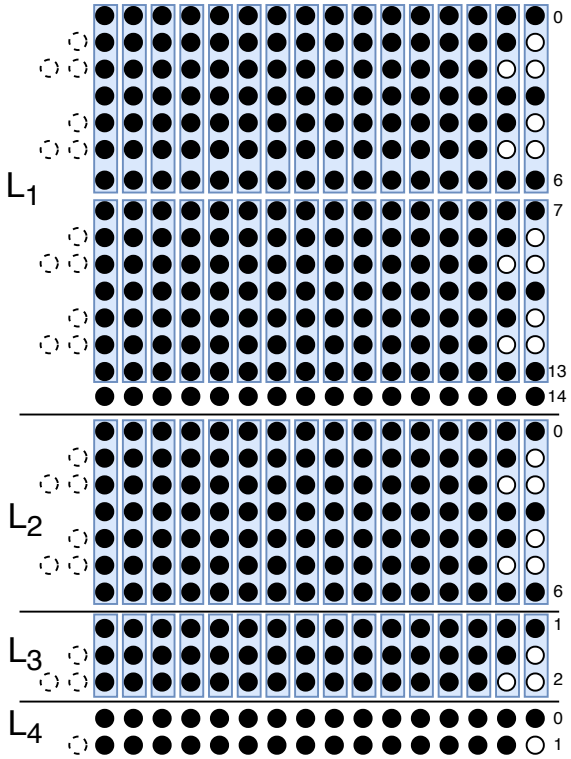
Re_{n-2} fills the hole produced by Re_1 and the overflow bits of Re_{n-1} fill the holes produced by Re_2. The second part (B) represents the same information, but it is arranged in a different way (Re_i is skewed). Then Row R_0 can be seen as the actual sum of a columns, R_1 represents the first carry and finally, R_2 contains the second carry. Noteworthy, using CSA the overflow bits do not need to be added again, because they do fit in the holes created by these adders. Consequently, the complexity is reduced, hence, an addition is removed.

Regarding the problem at hand, the first level of reduction is depicted in Figure 6.9(a). Each row is one of the thirty-three 16-bit words to be one's complement added. We have arranged the columns as follows: three clusters of 7-bit and two clusters of 6-bit all these clusters are reduced to 3-bit results. Consequently, 33 × 16-bit numbers are reduced to 15 × 16-bit numbers, as shown in Level 1 (L_1) of Figure 6.9(b). Observe that the white dots correspond to swinging the overflow bits from the dotted circles around.

The level 1 clusters two 7-bit elements per column, leaving one row for the next level, thus reducing from 15 to 7 numbers. In the second level (L_2), seven numbers are clustered and reduced to three numbers. As explained, with only three levels of logic, we are able to reduce 33 numbers to only 3 numbers without increasing the width of the numbers — 16-bit each. In what follow we discuss different alternatives to sum these three numbers in one's complement arithmetic to reach the final result.



(a) Reduction level 0.



(b) Reduction level 1, 2, 3 and 4.

Figure 6.9: Reduction tree data arrangement. (a) Level 0 of reductions. (b) Following levels of reductions.

Reduction Tree Version 1

The first version of the reducer tree, called *ArchRed1*, finishes L_3 with a ternary adder, plus two binary adders, as suggested in the following pseudo code.

```
L4       <= L3[2] + L3[1] + L3[0];
sumPrev  <=  L4(15:0) + L4(17:16);
sumFinal <=  sumPrev(15:0) + sumPrev(16);
```

Reduction Tree Version 2

The second version, *ArchRed2*, tries to reduce the three successive additions (L_3), processing in parallel the possible results and selecting the correct one with a multiplexer. The two most significant bits of the sum are used as selector.

```
L4[0] <= L3[2] + L3[1] + L3[0];
L4[1] <= L3[2] + L3[1] + L3[0] + 1;
L4[2] <= L3[2] + L3[1] + L3[0] + 2;
with (L4[0](17:16)) select
   sumFinal <= L4[0] when "00",
               L4[1] when "01",
               L4[2] when others;
```

Reduction Tree Version 3

The third version, *ArchRed3*, uses a 3 to 2 CSA generating a fourth level (L_4 in Figure 6.9(b)). Then, the two possible results are computed in parallel and a multiplexer selects the correct one. The most significant bit of the sum is used as selector. Using this solution only one 16-bit carry propagation is present in the critical path. In a Xilinx device, this means two CARRY8 components, therefore, yielding the best delay result.

```
L5[0] <= L4[1] + L4[0];
L5[1] <= L4[1] + L4[0] + 1;
with (L5[0](16)) select
   sumFinal <= L5[0] when "0",
               L5[1] when others;
```

Reduction Tree Version 4

A deeper look into the reduction tree reveals that two clusters in the L_0 are reducing only 6 words. Therefore, instead of feeding backwards a single word, which also includes a carry chain in the critical path. We tried feeding back level 3 to level 0 and convert the 6 to 3 CSA into 7 to 3 CSA. Hence, instead of adding 33 x 16-bit words each cycle we add 35 x 16-bit words, thus cutting down the critical path in the feedback. The reduction of L_4 is then needed only to get the final result. We call this implementation *ArchRed4*, Figure 6.10 depicts the reduction tree. This architecture is a little bit faster (1.8%) than *ArchRed3* but has more logic (13%), therefore, *ArchRed3* is a better option.

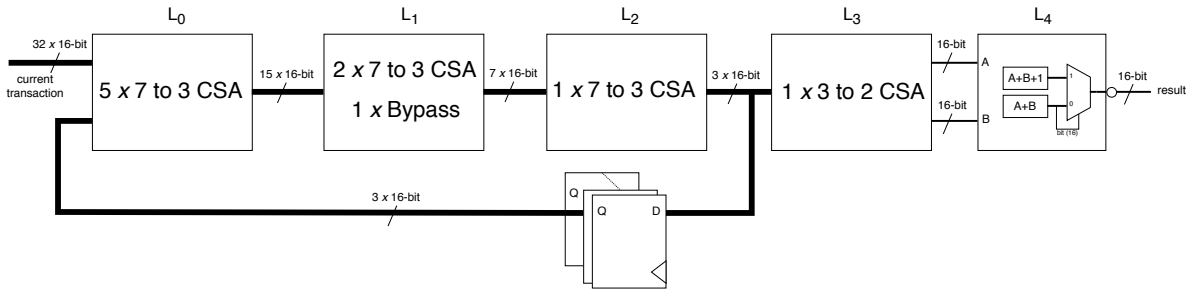


Figure 6.10: ArchRed4 low level reduction architecture.

6.7 Experimental Evaluation

All the previous architectures have been synthesized and implemented using Vivado 2017.4 for the Xilinx UltraScale+ architecture, targeting the VCU118 development board (section 3.8.1). Table 6.1 shows the delay and logic levels break-even for the studied circuits where *lgc* and *rt* stand for time spent in logic and routing respectively. Logic levels column details the components in the critical path. Additionally, the area and delay of the different architectures are presented. LUTs and CLBs usage are included, the amount of carry-logic component (carry8) is reported as well. The inputs and outputs were registered in order to obtain the post place and route timing report. The first element of *Max Delay* column summarizes the worst path expressed in *ns*.

Only reducer trees are the alternatives that meet timing at 322 MHz, the target frequency for the Xilinx's 100 Gbit/s Ethernet interfaces. The design *ArchRed4* is the one with the minimum delay. However, *ArchRed3* is only 1.8 % slower but 13 % smaller. What is more, the area penalty of *ArchRed3* is almost negligible compared to the naïve implementation.

The results shown in Table 6.1 regarding to delay are valid for Virtex UltraScale+ (16 *nm*), whereas for the Virtex UltraScale (20 *nm*) the same circuits have a delay penalty ranging from 27 % to 35 % due to the use of a previous generation node technology. Actually, in Virtex UltraScale, the best available design *ArchRed3* reaches only 3.7 *ns*. The best solution has been included as a part of Limago (chapter 7), implemented in the VCU118 board, the implementation of Limago reach similar delay results for the checksum logic as the ones shown in Table 6.1.

Figure 6.11 shows the performance of the checksum architecture depending on the message size. The 100 Gbit/s Ethernet link theoretical throughput is also shown. The plot demonstrates that the maximum possible performance is achieved comfortably even for the smallest packets. The figure also shows that the implementation reaches 165 Gbit/s when the message size is a multiple of 64-Byte — and, thus, there are no wasted bytes in the last AXI4-Stream transaction.

Circuit	Max Delay	Logic Levels	Area		
			LUTs	Carry8	CLBs
Bin16	4.684ns (lgc 2.4ns (52.1%) rt 2.2ns (47.9%))	20 (CY8=13 LUT2=6 LUT6=1)	541	98	117
Bin32	5.278ns (lgc 2.9ns (54.6%) rt 2.4ns (45.4%))	25 (CY8=17 LUT2=7 LUT3=1)	530	85	103
Tern16	4.073ns (lgc 2.0ns (49.8%) rt 2.0ns (50.2%))	17 (CY8=11 LUT2=2 LUT3=3 LUT6=1)	368	53	96
Red1	3.691ns (lgc 1.6ns (42.9%) rt 2.1ns (57.1%))	13 (CY8=6 LUT2=1 LUT6=4 MUXF7=2)	707	8	130
Red2	3.070ns (lgc 0.9ns (29.0%) rt 2.2ns (71.0%))	11 (CY8=3 LUT5=1 LUT6=4 MUXF7=3)	748	5	138
Red3	2.979ns (lgc 0.9ns (31.5%) rt 2.0ns (68.5%))	10 (CY8=2 LUT3=1 LUT5=1 LUT6=3 MUXF7=3)	734	5	140
Red4	2.925ns (lgc 0.955ns (32.650%) rt 1.970ns (67.350%))	x (CY8=3 LUT3=2 LUT6=3 MUXF7=3)	831	5	188

Table 6.1: Delay and logic depth break-even for the studied circuits in an UltraScale+ architecture.

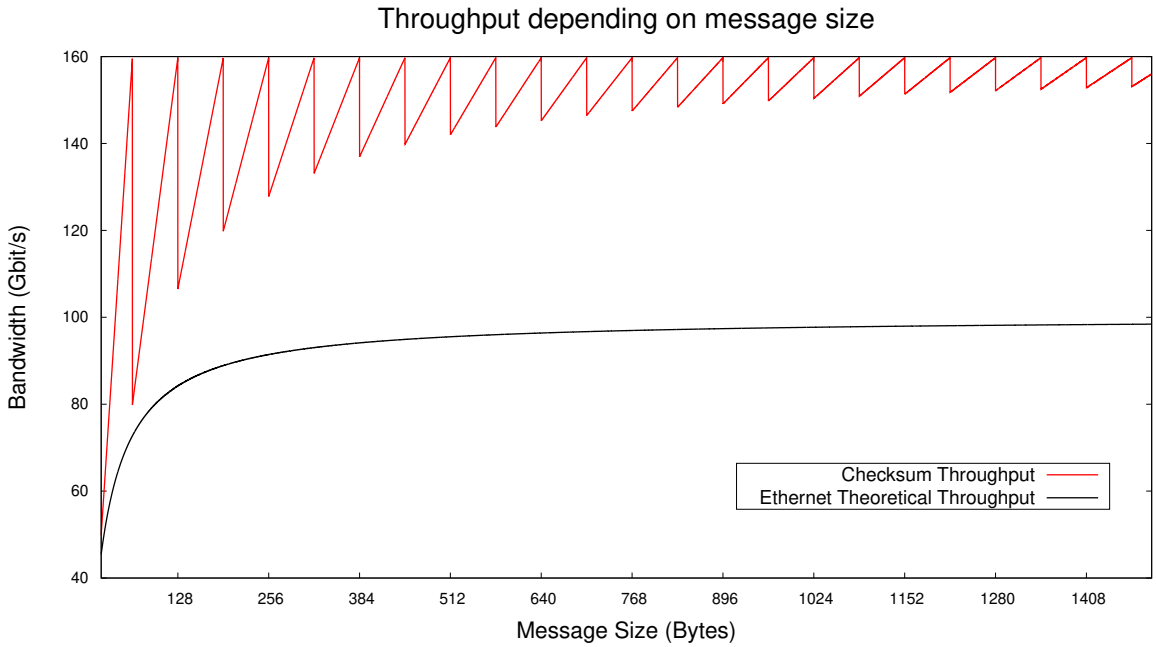


Figure 6.11: Checksum computation performance.

To support a hypothetical 200 Gbit/s link, we assume that the bus width will double to reach a 1024-bit AXI4-Stream. In such a case, the checksum for TCP/UDP in the worst-case scenario can be reduced to 65 × 16-bit word one's complement addition at 322 MHz. Following the same idea as described in the design *ArchRed3*, with an extra level of reduction, it is feasible to reach the required processing rate. However, the delay of this new level has to be cut out from the routing. In the current Virtex UltraScale+ architecture this is only viable with a careful relative placement of the logic. In conclusion, the throughput of the proposed design can be doubled, but further work is needed to do so. Figure 6.12 shows how the low level architecture would look like for a hypothetical 200 Gbit/s implementation.

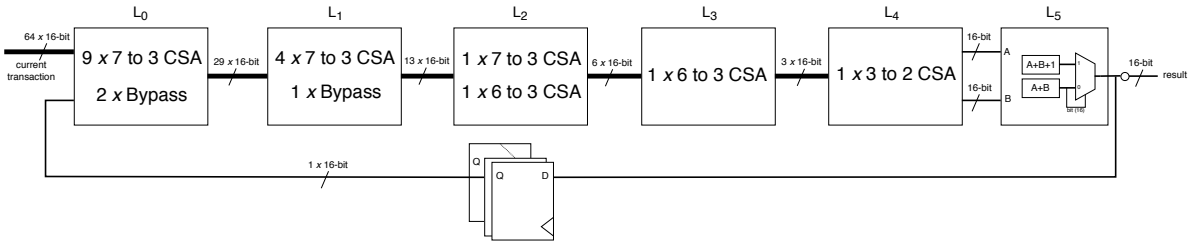


Figure 6.12: Possible architecture for 200 Gbit/s checksum computation leveraging CSA.

6.8 Conclusion

In this chapter we have shown how to efficiently calculate the one's complement checksum in a 100 Gbit/s Ethernet links taking advantage of the Xilinx Virtex UltraScale+ architecture and CSAs. The result of this chapter are published on the paper [148].

We were able to reduce the one's complement checksum as the addition of 33 × 16-bit numbers in one's complement arithmetic at 322 MHz. After evaluating several alternatives, we conclude that the best solution is implemented using tree levels of 7 to 3 CSA, which fits well in the slice Xilinx's devices architecture. This means that, after three levels of logic, the problem is transformed in the one's complement addition of 3 × 16-bit numbers. For the final addition a 3 to 2 CSA adder, two binary adders in parallel and a multiplexer are used. This architecture reaches the desired throughput with a negligible area penalty in Virtex UltraScale+ devices, achieving an initiation interval of one and one clock cycle of latency, extremely useful to reach maximum throughput with short packets. Extrapolating such idea, a checksum offloading engine at 200 Gbit/s is feasible with a meticulous relative placement of the logic. All the designs discussed are available as open-source [149].

Achieving similar results with HLS is not feasible due to the lack of low level optimization available in such a tool. However, this is not necessary damaging because the traditional HDL fulfill such purpose and the integration between HLS and HDL modules can be done easily using standard interfaces. What is more, since Vivado-HLS 2019.1 the RTL blackbox concept has been introduced, where a user can integrate RTL code within the tool making the integration even simpler.

Finally, we use the result of this chapter to implement a 100 GbE full TCP/IP stack. Limago is discussed in the next chapter chapter 7.

CHAPTER 7

RELIABLE DATA TRANSMISSION

The realization that the network is becoming an important bottleneck in computing clusters and in the cloud has led in the past years to an increase scrutiny of how networking functionality is deployed. From TCP Offload Engines to Software Defined Network, including Smart NICs and In-Network Data Processing, a wide range of approaches are currently being explored to increase the efficiency of networks and tailor its functionality to the actual needs of the application at hand. To address the need for an open and customizable network stack, in this chapter we introduce Limago, an FPGA-based open-source implementation of a TCP/IP stack operating at 100 Gbit/s. To our knowledge, Limago provides the first complete description of an FPGA-based TCP/IP stack at these speeds, thereby illustrating the bottlenecks that must be addressed, proposing several innovative designs and showing how to incorporate advanced protocol features into the design without jeopardizing throughput. As an example, Limago supports the TCP Window Scale option, addressing the Long Fat Pipe issue, which arises in very high-speed connections. Limago not only enables 100 Gbit/s Ethernet links in an open source package, but also, it paves the way to programmable and fully customizable NICs based on FPGAs harnessing HLS.

7.1 Introduction

The growing amount of data and the complexity of the workloads that characterize modern distributed computing have turned the network into a potential bottleneck [3].

Besides, in cloud environments, the network also limits the number of virtualized/containerized applications that can be deployed on a single server: The more CPU cycles needed to deal with an increasingly complex networking stack — which needs to provide not only TCP/IP packet processing but additional functionality such as Network Function Virtualization (NFV) or Remote Direct Memory Access (RDMA) — the less CPU cycles that are available to applications. In addition, the trend towards specialization seen in cloud computing opens up the possibility of tailored network designs through Smart Network Interface Cards (NICs), which push application-level processing to the network [150]. As a result, we are witnessing a flurry of activity around programmable networks based on a variety of designs and architectures.

Data processing in distributed applications often requires large data transfers between machines. In these applications, low bandwidth or inefficiencies in the network processing can significantly reduce the overall performance. In recent years the bandwidth of commodity Ethernet in data centers has increased to 40 Gbit/s and is transitioning towards 100 Gbit/s. To cope with the increasing network bandwidth and reduce the CPU cycles spend on network processing, network cards have been adding increasingly more offloading capabilities, *e.g.*, checksum calculation, TCP segmentation, or Receive Side Scaling (RSS). High-performance network cards are even able to offload complete network stacks in the form of TCP Offload Engine (TOE). Despite these efforts, I/O inefficiencies and data movement overheads remain, this effect is even worse at a higher network bandwidth a situation worsen as the increasing network bandwidth requires the CPU to process data at a higher rate.

On the other hand, it is well-known that TCP/IP is a complex protocol that burdens the CPU. In the literature [151–154] there is a general agreement on, (i) TCP/IP is a demanding protocol, as a rule of thumb to process 1 b/s of TCP/IP it is necessary 1 Hz of CPU. (ii) Dedicated hardware improves significantly the TCP/IP performance. (iii) However, developing an ASIC is not advisable because it cannot cope the update in the TCP/IP protocol and it is not suitable for every application. In this light, FPGAs emerge as the optimal solution to implement TOE either as a host attached card, or even as a standalone solution (network-attached).

An industrial grade example of these developments is provided by Microsoft Catapult [155], a deployment of FPGAs in the cloud that has evolved through several generations [156, 157]. The current version inserts an FPGA on the data path between the top of rack switch and the server machine, bump-in-the-wire. Hence, all network traffic in and out of the host goes through the FPGA. The FPGA is then used to augment the network functionality with system and application-level features. For instance, it can be used as a customizable smartNIC to offload network virtualization functionality [157], application-level functionality such as RDMA packet processing to support key-value

stores [158], or for distributed machine learning algorithms [159]. For inter-FPGA communication, Catapult uses a proprietary protocol called Lightweight Transport Layer (LTL) [155]. LTL provides a connection based interface on top of UDP by introducing packet sequence numbers and acknowledgments to guarantee reliability and packet ordering.

Catapult is, by far, not the only possible design. In IBM's cloudFPGA [22], the FPGA is deployed as a network-attached accelerator. Similarly, Caribou [31] deploys FPGAs as storage nodes that extend the TCP/IP stack with distributed consensus functionality (a network function) [160] as well as scans and string processing (application-level functionality) [31, 161]. What is more, Sapio *et al.* [162] discuss the opportunities and challenges of in-network processing, FPGAs suit very well this offload paradigm. In this context, Tokusashi *et al.* have recently explored in-network processing [150] comparing FPGAs, programmable switches, and ASICs. These examples have shown that an FPGA-based NIC can accommodate a tight integration of networking and accelerator logic, providing high throughput processing at low average and tail latencies. The latter is especially challenging to achieve on a CPU-based system.

Promising as they are, for FPGA-based designs a challenge remains: scalability with increasing network bandwidth. To address this challenge, we introduce Limago, an open-source 100 Gbit/s TCP/IP network stack on an FPGA. Limago explores the changes needed to upgrade an existing open-source TCP/IP stack from 10 Gbit/s [30] to 100 Gbit/s, but maintaining the same high-productivity design methodology, based on Vivado-HLS, which was utilized in the previous design. In doing so, Limago illustrates how to tackle the problem of FPGA-based packet processing at such rates. From the existing design, Limago inherits the scalability in terms of the number of connections as well as the control flow and congestion avoidance functionality. Limago not only transforms and adapts these existing features to increase the supported bandwidth from 10 Gbit/s to 100 Gbit/s, but also contributes novel features widening its applicability. The changes are non-trivial extensions of the existing stack. For instance, the data path had to be widened eight times and the operating frequency doubled to reach the target bandwidth, several low-level architectural changes and balanced pipeline stages were necessary to meet timing, and accessory modules were redesigned so as to match the current needs. Limago also incorporates functionality such as the TCP *Window Scale* option, an extension to the basic TCP/IP protocol which addresses the *Long Fat Pipe* issue section 7.2.1.

Limago serves as a platform for further research in programmable networking and as a design guideline on how to tackle high network bandwidths with FPGA-based systems. The key contributions of Limago are:

- First open-source FPGA-based 100 Gbit/s TCP/IP stack.

- Very low communication latencies below 1.5 μsecond.
- Full *Window Scale* support — 64 KB to 1 GB.
- Fully developed using High-Level Synthesis (HLS), making it easier to modify and maintain, apart from the checksum computation.
- High data transfer for multiple connections.
- Customizable option at synthesis level.

7.2 TCP/IP Background

Computer network is organized in seven layers, according to the Open Systems Interconnection (OSI) model [163]. From the lowest layer (layer 1, the physical layer controlling access to the physical medium) to the top layer (layer 7, or application layer), each layer abstracts the details of the level below and provides a well-defined interface to the level above. Figure 7.1 shows the standard OSI layers, including their names and numbers. The TCP/IP is a suite of communication protocols covering layers 3 (Network layer, IP) and 4 (Transport layer, TCP) addressing how data should be split into packets, addressed, transmitted, and received at the destination. TCP/IP is widely used both in the Internet as well as in data centers. What is more, TCP/IP is the core of the Internet as we know it today.

TCP provides a reliable data transmission using a connection-oriented mechanism between two end points. With TCP, the two communicating parties have to establish a

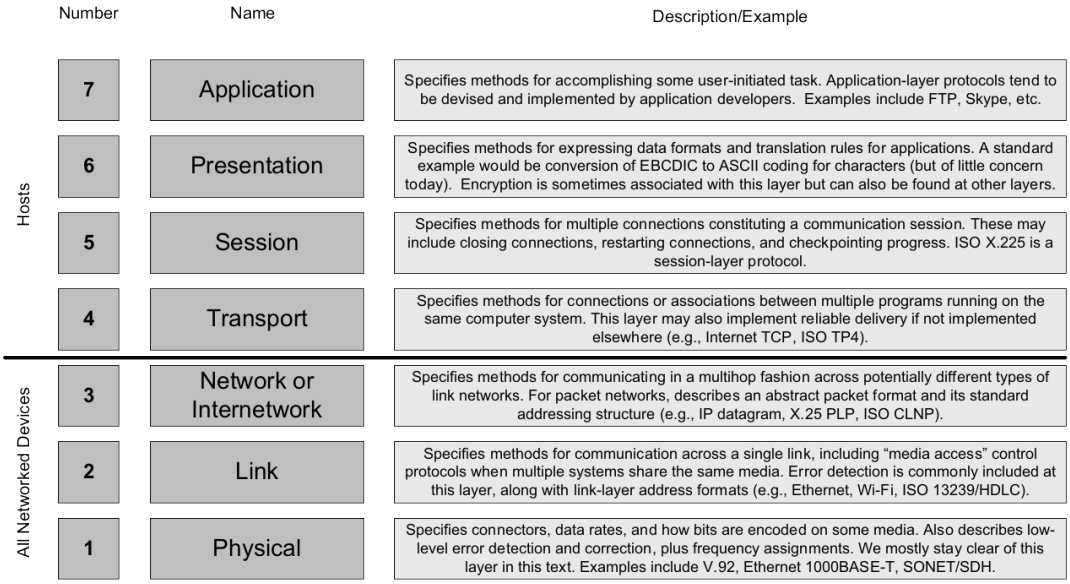


Figure 7.1: The standard seven-layer of the OSI model. Source: [164].

connection before data can be exchanged. TCP converts the data sent by the application into a set of packets that the IP (layer 3) can transport. This action is called packetization. Each packet contains a sequence number, representing the offset to the first sent byte. This feature allows packets to have variable size. The TCP implementation determines the best size for the packet to be sent, commonly fitting each segment into a single IP packet in order to avoid fragmenting an IP packet.

RFC793 [135] describes the original implementation of TCP. In what follows a brief description is provided. There are three separated parts in a TCP connection. Before data can be exchanged a connection has to be established in a process known as **three-way handshake**. Let us say that we have two machines A and B. Machine A sends a SYN packet (identified as such by one of the flags in the header), with the port number, a random sequence number. If machine B is ready to communicate in that port, a SYN-ACK packet is sent to machine A. The SYN flag says that the connection is being opened and the ACK flag acknowledges the first SYN packet. Finally, to acknowledge the SYN-ACK packet, an ACK packet is send from machine A to machine B. At this point the connection has been established and data can be exchanged in any direction.

For the actual **data exchange**: each packet has a sequence number (SEQ), which is the offset to the first sent byte. Each transmitted packet has to be acknowledged (ACK), with packets being re-transmitted after a certain time-out until the ACK is received. As a result, each packet is stored until acknowledged in case a re-transmission is needed. The process determining which packets are in flight, which ones have been received or are waiting for an acknowledgment is called sliding window protocol because of the circular buffers involved at the sender and receiver to keep track of what has been sent, what has been received, and what needs to be or has been acknowledged. This method of transmission would be inefficient if the sender waits to transmit a new segment until the previous segment has been acknowledged, the channel will be idle most of the time. To maximize bandwidth, the sender transmits many consecutive segments, up to the receiver's advertised window — this determines flow control mechanisms based on the dynamic re-sizing of the sliding window, which is a defined in 16-bit field (64 KiB) on the TCP header. This field defines the number of bytes the recipient can hold. The sender cannot have more than that number of bytes pending acknowledgment, otherwise packets will be lost. The recipient can acknowledge several segments in a single message, also saving bandwidth by doing so, namely, delayed acknowledgment. When all data have been transmitted, FIN packets are sent in both directions and the connection ends, known as **teardown**. Figure 7.2 shows the three-way handshake and teardown TCP process— no data is exchanged in such example. The book [164] provides an in depth analysis of the TCP standard as well as the other network protocols.

TCP/IP stack is known to be very memory intensive, due to the large amount of

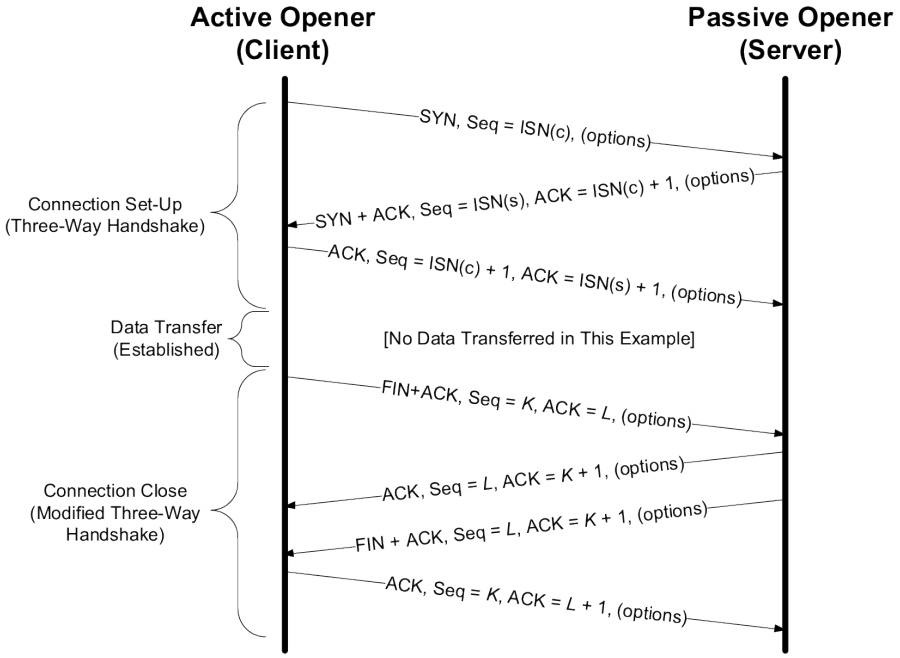


Figure 7.2: A normal TCP connection establishment and termination. Source: [164].

memory needed to keep track of the current state of each connection. What is more, multiple connections are supported by the same machine, consequently, TCP/IP is a very challenging stack.

7.2.1 Long Fat Pipe

TCP is widely used and, for that reason, many extensions and optimizations have been proposed. One such optimization addresses the problem of the so-called *Long Fat Pipes*. The problem arises when the $RTT(s) \times LinkCapacity(bit/s) > BufferSize(bit)$, at which point the channel capacity never will be reached due to pauses in the transmission. The reason is simple; the sender cannot transmit more data than the receive buffer can hold. More specifically, the amount of unacknowledged data in-flight is limited to the size of the receive buffer. In the original implementation, the buffer size (window) is limited to 64 KiB as specified by the 16-bit field in the TCP header. To tackle this issue, the Window Scale option was introduced [165]. This TCP option negotiates a Window Scale (WS) factor during the three-way handshake. This scaling factor determines the buffers size through the following equation: $64KiB \times 2^{WS}$, enabling buffers size to growth of up to 1 GiB, 128 times bigger than the original implementation. This TCP option is backwards compatible. If one of the end-point does not support it, the scaling factor will be set to zero thus, not scaling. What is more, in the negotiation phase the window scale is set to the minimum value advertised by the end-points.

7.3 Challenges at 100 Gbit/s

Limago uses the 322 MHz clock provided by the integrated 100G CMAC, and a 512-bit AXI4-Stream interface (section 3.8.1). With respect to the 10 Gbit/s version, that is an eight times increase in the width of the datapath and more than a two times increase in the operating frequency. Moreover, the smallest packet (64-Byte) just fits into a single transaction and, for such short packets, the processing rate must be 148.8 million packets per second. The greater data rate implies novel designs for several components often taken for granted. For instance, existing SmartCAM [166] design, used for flow identification, do not operate at such frequency and a new solution is thus needed, what is more, the previous design is not open-source. Similarly, certain optimizations are optional at lower rates, but a must at such bandwidth. For instance, the *Long Fat Pipe* issue might not be observable at 10 Gbit/s but must be addressed to reach 100 Gbit/s. This requires additional circuitry to support and negotiate the TCP Window Scale option. Furthermore, the computation of one's complement checksum poses a real challenge at such speed.

7.3.1 TCP/IP Checksum

Checksum computations are widely used when processing TCP/IP packets. Limago uses an efficient implementation, leveraging 7 to 3 Carry Save Adder (CSA) circuits [147] to calculate the checksum within one clock cycle. The module was written in HDL to achieve a low-latency in this recurrent circuit. Actually, this is one of the few modules of Limago written in HDL; the vast majority of blocks are written in Vivado-HLS. But in this case, an efficient and low-latency implementation was needed, impossible to achieve with the Vivado-HLS version being used (2018.2). The circuit is described in detail in chapter 6 and in [148].

7.3.2 CuckooCAM

The 10 Gbit/s version of the stack uses the smartCAM [166] module provided by Xilinx as a fast lookup engine, which is not open-source. Such module uses a four-tuple consisting of IP source and destination addresses plus TCP source and destination ports as a key, 96-bit key. We replaced this module with our own implementation, CuckooCAM, based on cuckoo hashing and providing an initiation interval of one clock cycle for lookup and deletion. In CuckooCAM, insertion time depends on the load factor and the occupancy rate can exceed 90% due to a secondary memory structure known as a stash. It is clocked at 322 MHz, providing 322 million lookups per second — this lookup rate is achieved when there are no insertions. The width of the key and value are configurable; therefore,

we have reduced the size of the key to a three-tuple by removing Limago's own IP address which does not change during operation, consequently we use a 64-bit key. The reduction of the key from 96-bit to 64-bit results in a significant reduction in BRAM usage for this module (22%) — this result is for an implementation that supports more than 10,000 elements. We were able to develop such a demanding design very quickly using HLS with exceptional Quality of Result (QoR).

7.3.3 DRAM Memory Access

To support a large number of connections, the TOE uses external memory for its receive and send buffers. In particular, this is necessary for the send buffer, where the payload has to be stored until it is acknowledged — at least for a period of time equal to RTT. DRAM memory bandwidth has not increased at the same exponential rate as the Ethernet links [167], therefore, we need to assess if the off-chip memory provides enough bandwidth to fulfill Limago's requirements. For 100 Gbit/s, the resulting requirements in terms of memory bandwidth are close to the peak bandwidth provided by DDR4 present in the VUC118 board (section 3.8.1). To reach such bandwidth, we use sequential access as much as possible, *i.e.*, the bigger the segment size the better. Additionally, the offsets into the receive and send buffer are determined by the TCP sequence number. This can result in unaligned memory accesses affecting the memory bandwidth further. Therefore,

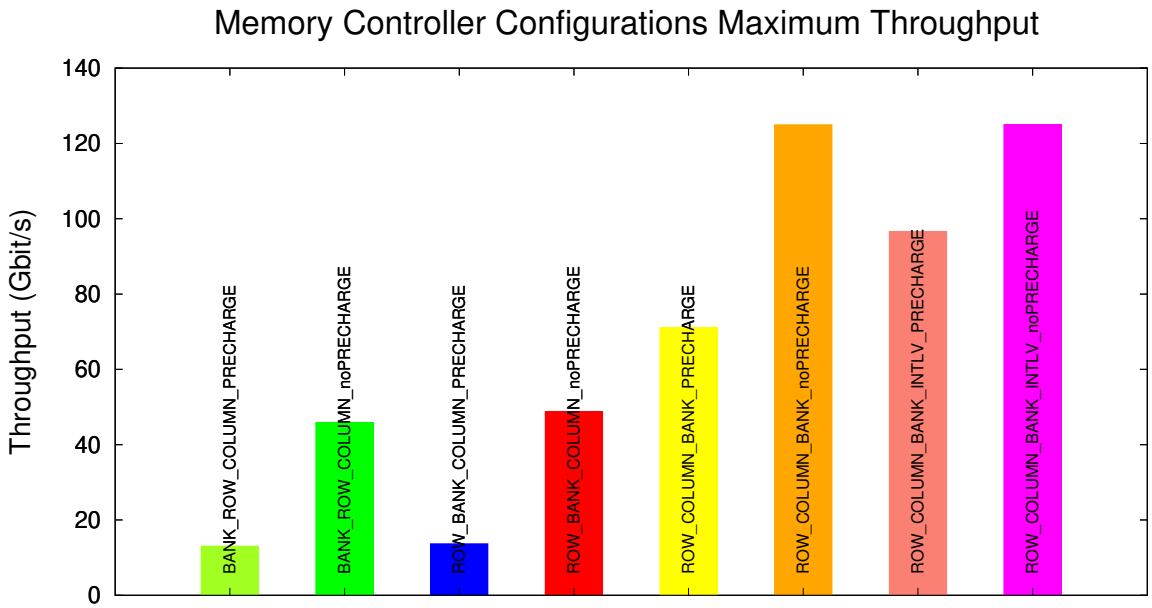


Figure 7.3: Memory controller maximum performance with different memory address map, sequential write.

we verified the viability of storing the buffers in the external DDR4-2400 through several microbenchmarks, varying the memory-alignment as well as the access size. First of all, we evaluate which address map adapts better for sequential writes, Figure 7.3 shows a summary of the experiments. The plot also shows that, for our scenario the precharge option worsen the performance. The two best options are using ROW_COLUMN_BANK with and without interleaving. We observed a peak bandwidth of 125 Gbit/s with 64-Byte aligned words and approximately a 6% performance loss when transfers were not aligned, thereby ensuring the design achieves enough memory bandwidth for all cases. Since the buffers in external memory are organized as a circular buffer, additional logic is required to handle the wrap-around when the "end" of the buffer is reached. Particularly, a single data transfer is split into two transfers (one before the wrap-around and one after), requiring data re-alignment. The HLS code for this module was redesigned carefully to guide the synthesis tool to the most efficient implementation involving a 64 to 1 multiplexer.

7.3.4 TCP Window Scale Option

Links with a large $bandwidth \times delay$ product suffer from the *Long Fat Pipe* issue (section 7.2.1): those links where the bandwidth × delay product is larger than the buffer size [165]. The Window Scale option is used to allocate any fix-size buffer in the range of 2^{16} to 2^{30}-Byte, thereby leading to a better usage of links — reducing or eliminating the idleness in the link.

Currently, Window Scale is the only supported TCP option in Limago. Due to the lack of a standard TCP option layout, the parsing of options is done sequentially, one clock cycle each. Fortunately, the Window Scale option is only negotiated during the initial three-way handshake, for instance, options are only parsed once in the lifetime of a connection. The Window Scale is set to the minimum value advertised by both endpoints. The buffer size is automatically set to the negotiated value, which can never be bigger than the one announced. The flow control also benefits of this option, because more bytes can be sent consecutively. Support for the Window Scale option has to be enabled at synthesis.

Finally, the maximum number of connections depends on the external DRAM capacity and the Window Scale factor, as shown by Equation 7.1. $DRAM_b$ is the $log_2(DRAMSize)$ and WS_b is the $log_2(WindowScale)$. As an example, with 4 GB of DRAM and a Window Scale of 128, $2^{32-7-16} = 2^9 = 512$ concurrent connections can be supported. In this example, only the *Tx Buffer* is taken into account, if the *Rx Buffer* is enabled, the number of connection is reduced in half unless there are two independent memories for each buffer, which for 100 Gbit/s is almost mandatory to achieve maximum throughput.

$$\#conn = 2^{DRAM_b - WS_b - 16} \tag{7.1}$$

7.4 Related Work

The benefits of TCP Offload Engine (TOE) are well-known [168–170]: reduced CPU utilization and bypassing of the Operating System. In a TOE, packet processing is moved to the NIC, whereas the control decision remains in the host. Nowadays, most NICs offer some degree of offloading. In this section, we focus on FPGA implementations of TCP/IP stacks.

LDA technologies [171] offers an ultra-light, ultra-high speed and ultra-low latency (20 ns) FPGA-based TOE. Their solution includes independent transmitter and receiver modules. Each module can handle thousands of connections and an external memory is not necessary. For sixteen connections, 44 BRAMs and 2,704 LUTs are necessary. Neither the implementation details nor the maximum throughput are available, but, published results for this TOE using Solarflare NICs are based on 10 Gbit/s connections. Chevin Technology [172] offers a 10/25 Gbit/s TCP/IP core, which can work both as client or server. It supports up to 256 simultaneous connections. The Tx and Rx buffers can be configured from 1 KiB to 1 GiB, implying Window Scale support. For sixteen connections, 5 BRAMs and 12,000 LUTs (plus the external buffer) are necessary. Enyx [173] offers an RTL TOE solution with up to 4,000 connections, but not further details about resource utilization are provided. They also have announced a 25 Gbit/s implementation [174]. Dini [175] offers a 10 Gbit/s solution where the FPGA is used as a NIC. The buffer size is configurable from 4 KiB to 64 KiB and it supports up to 128 connections per instantiated IP-Core and out-of-order packet delivery. Algo-Logic [176] supports full duplex rates up to 20 Gbit/s per instance, claiming more than 200 Gbit/s can be achieved with multiple instances. The design targets low-latency applications such as high-frequency trading.

The authors in [177] presented a comparison of three 10 Gbit/s alternatives: a pure software TCP/IP stack, a software TOE with kernel-bypassing and a hardware TOE (Fraunhofer HHI 10 GbE TCP/IP) with kernel-bypassing, concluding that the hardware solution has less latency and a more deterministic behavior. The work in [178] presents a complete TOE implementation supporting jumbo frames and configurable Maximum Segment Size (MSS) and timestamp. Only one connection is supported with a 90 ns latency for a 100-Byte packet. Their solution is compared against a commercial, one achieving better latency. Bianchi *et al.* [179] introduce a TCP implementation using XFSMs, which is claimed to be “code-once-port-everywhere”. The implementation is tested over three different architectures, software, FPGA, and NS3 emulator [180],

reaching similar results. At CERN [181], a simplified and unidirectional 10 Gbit/s TCP/IP implementation was made for a lossless data collection using an FPGA. Probably, the closest work to Limago is [153], an asymmetrical standalone TCP/IP implementation oriented to video-on-demand, which supports 20,480 connections working as a client and 2,048 connections working as a server. It also can send up to 40 Gbit/s but only receive up to 4 Gbit/s. The starting point for Limago is a 10 Gbit/s TOE written by Sidler *et al.* in C++ using Vivado-HLS [30, 144]. Such design targets the Virtex 7 FPGA family architecture.

7.5 Limago Architecture

The main focus of this chapter is how to implement an efficient 100 Gbit/s TOE using HLS as much as possible. To do so, layers 1 and 2 of the OSI model have to be addressed as well. In this section, we dive deep into our framework which is used along with the TOE to provide the full network stack capabilities, part of this infrastructure may also be used as a shell in other designs. The first version of Limago runs on a Xilinx's Ultrascale+ VCU118 development board (section 3.8.1) and we also have implemented Limago for the ALVEO U200 board (section 3.8.1).

Figure 7.4 shows Limago's main components. We use AXI4-Stream to interface with the application logic as well as with the network modules. Since CMAC exposes an LBUS interface, we added a custom adapter module that converts between AXI4-Stream and LBUS (section 3.8.1)— since Vivado 2019.1 the IP-Core includes AXI4-Stream optional support. This piece of hardware was done using Verilog to have total control even for short packets.

Rx and Tx checksum are shortly introduced in section 7.3.1 and detailed in depth in chapter 6. Additionally, **CuckooCAM** is briefly presented in section 7.3.2, its implementation details fall out of the scope of this thesis. In what follows the rest of the components of the infrastructure are described in detail, all of them have been developed using HLS unless stated otherwise.

Inbound Packet Handler: parses Ethernet and IPv4 headers of every incoming packet and determines to which category it belongs. Currently, the following kind of packets are supported: Address Resolution Protocol (ARP), Internet Control Message Protocol (ICMP), TCP and UDP. If the packet matches the filter, the signal TDEST will carry a different identifier for each kind of packet. Then an AXI4-Stream Switch forwards the packet to the appropriated module. If the packet does not belong to one of the previous categories, it is dropped. In addition, the Ethernet header is removed for ICMP, UDP and TCP packets and the rest of the packet is realigned.

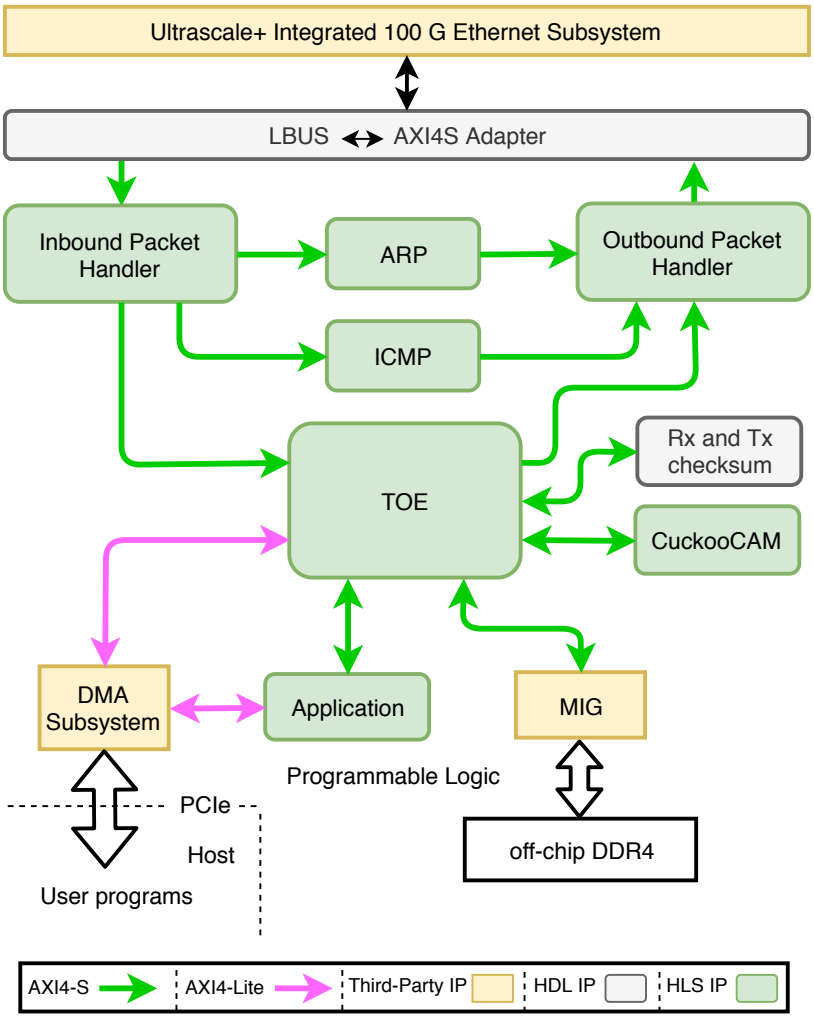


Figure 7.4: Limago general architecture overview.

ARP module: is in charge of the ARP packets. When an ARP request arrives and the IP address matches, it generates an ARP reply packet. Its main function is to associate IP addresses with Media Access Control (MAC) (physical) addresses, which is done using a 256-element table. The ARP module also receives MAC address requests from the *Outbound Packet Handler*. If the entry is not present in the table, an ARP request packet will be generated, and a miss will be reported. Additionally, at system start up, an aggressive MAC discovery address is implemented — an ARP request is sent to every IP address within the range of the subnet (LAN) in order to obtain the associated MAC address. In doing so, we avoid of retransmitting the first packet of each connection.

ICMP module: provides responses to echo request packets, *a.k.a.*, ping. The module is useful to verify connectivity and gives a fair estimation of the Round-Trip delay Time (RTT).

Memory Interface: is composed of a Data Mover and Memory Interface Generator

(MIG), both Xilinx IP-Cores. The MIG exposes a 512-bit AXI4 memory mapped interface and communicates with the off-chip DDR4 memory. The Data Mover is in charge of merging data and commands, which are produced in a streaming fashion from the TOE module, to an AXI4 interface. The maximum frequency of the MIG is limited to 300 MHz, a theoretical maximum of 153.6 Gbit/s could be achieved, but that is enough to transfer the payload to the external memory (section 7.3.3). The clock domain crossing is done using the internal FIFO of an AXI4-Interconnect.

Outbound Packet Handler: gathers packets coming from ARP, ICMP and TOE modules. ARP packets are forwarded directly. However, ICMP and TCP packets need to be prepended with the Ethernet header (14-Byte). If needed, a MAC address lookup, consisting of the IP destination address, is issued to the *ARP module*. If the lookup is a hit, the Ethernet header is constructed using the returned MAC address, prepended to the packet, and transmitted. Otherwise, the packet is dropped and an ARP request is generated instead. Moreover, the packet size is evaluated and padded to 60-Byte if needed, to comply with the CMAC specifications.

DMA subsystem: we use the DMA for PCI Express (PCIe) Subsystem Xilinx IP-Core for providing users access to memory mapped registers within the logic. Limago uses the Xilinx's drivers both for debugging and communication. The necessary customization layers are built on top of them. There is another debug module, JTAG2AXI, when there is not PCIe connectivity. It fulfills the same need, debug or access to the current state of the design.

7.6 TOE Architecture

This section describes the overall architecture of the TOE (Figure 7.5). As a starting point we used an open-source 10 GbE TOE implementation [30], because it is written in C++, it is scalable in terms of concurrent connections and it has been proved to give good performance. The internal structure is divided into three parts, the incoming data path (*Rx Engine*), the outgoing data path (*Tx Engine*), and the state-keeping data structures. The figure also shows the external buffer *Tx and Rx Buffer*, these buffer use off-chip memory, in Limago the *Rx Buffer* is optional. Furthermore, the dash boxes are optional modules that can be enabled at synthesis. We also use HLS to implement this module, however, we made few optimizations in the code to guide the tool to a better result.

7.6.1 Rx Engine

Incoming packets are processed by the *Rx Engine*. To verify the checksum, first the TCP pseudo header is constructed and prepended to the packet payload (Figures 6.3 and

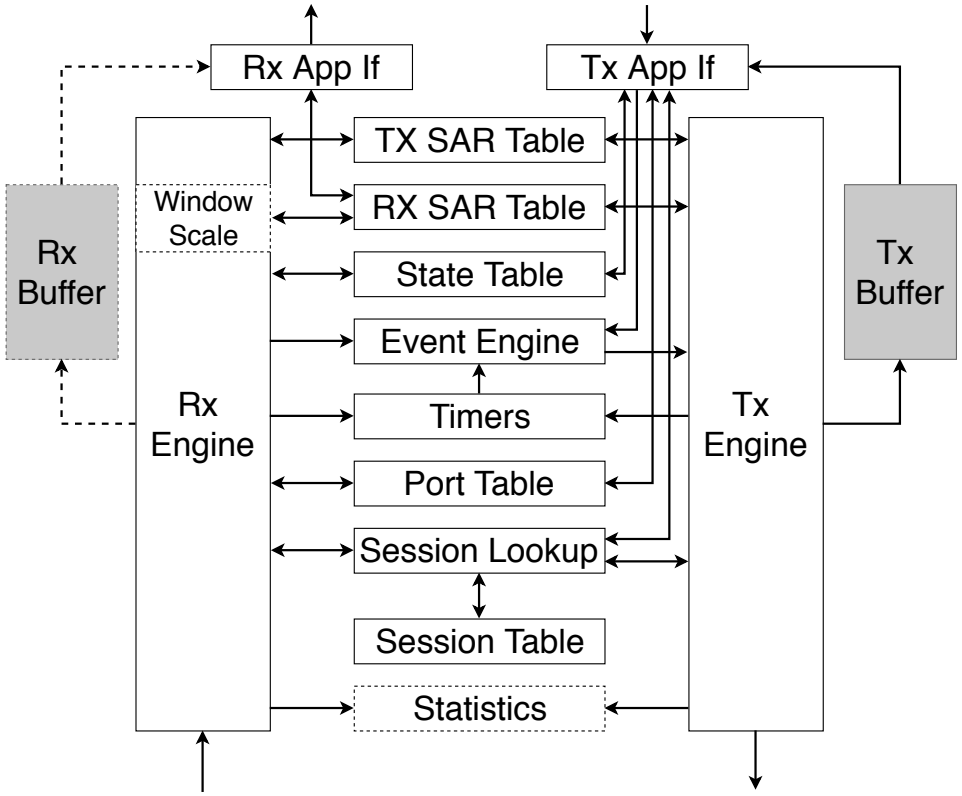


Figure 7.5: TOE architecture overview.

6.4 respectively). The pseudo header and packet payload are then forwarded to the *Rx checksum* module. If the result equals to zero, the checksum is valid, and the packet is passed on to the next module, otherwise it is dropped. Valid packets are parsed to extract the necessary fields from the IPv4 and TCP headers, which is done in one clock cycle, at the same time the pseudo header is removed. The *Rx Engine* contains a Finite State Machine (FSM) that takes decisions based on the extracted fields. First, it looks up the destination port in the *Port Table*, if the port is not in the `LISTEN` mode the packet is discarded. Next, using the three-tuple — IP source address, TCP source and destination port — a look-up to the CuckooCAM is issued. The CuckooCAM returns a tuple with a boolean flag indicating if the lookup is a hit, and a 16-bit sessionID. The sessionID is used as an index to look-up the state of the connection in all the other tables. If the lookup was a miss but the packet has the SYN flag set, the three-tuple is inserted with a new sessionID and a SYN-ACK event is generated. The FSM uses the sessionID to retrieve the sequence and acknowledgment number from the two *SAR Tables* and, if necessary, updates them. Finally, if the packet contains a payload, a notification is sent to the application while the payload is written to the *Rx Buffer* — in the case that is enabled. The FSM in the *Rx Engine* enforces a strict order of the packets and currently does not support out-of-order processing.

7.6.2 Data Structures

Session Lookup: provides the means to interface with the CukooCAM, using the three-tuple to obtain the sessionID. It handles the requests from the different producers and forwards the response properly. The sessionID is used to index every data structure to access the state of the corresponding connection. For each incoming packet a lookup is issued to the CukooCAM module implemented outside this module (section 7.3.2). In case of a SYN or a SYN-ACK packet, if the three-tuple has not been inserted yet, it will be inserted using a new sessionID identifying the new connection. Additionally, the *Session Lookup* module contains a table that maps the sessionID to the three-tuple. This mapping is used by the *Tx Engine* to generate the IPv4 and TCP headers of outgoing packets.

Port Table: keeps track of the state of each port, which can be `CLOSE`, `LISTEN` or `ACTIVE`. The standard port range for static and ephemeral ports are used. It is queried for every incoming packet in order to check the state of the port. If an incoming packet targets a port in `CLOSE` state, it is discarded and a RST packet is generated as a response.

State Table: stores the current state of each connection as specified by RFC793 [135]. To simplify the implementation, the CLOSE and LISTENING states are merged. The *State Table* can be updated by the *Rx Engine* when incoming packets are processed and by the *Tx App If* when the application opens a new connection. Consistency is guaranteed by using atomic operations. The locking is fine-grained so that only the currently accessed entry is locked.

Timers: this module supports all time-based event triggering as required by the protocol, three timer modules are implemented: *Re-transmission*, *Probe* and *Time-Wait Timer*. It follows the same approach of the original version, using the approach introduced by [182], which provides linear scaling of on-chip memory. This is a viable approach, as TCP timers operate with a millisecond granularity. Given a clock period of 3.103 ns and one timer access per clock cycle, we can update 322,268 connections per millisecond.

Event Engine: gathers events from the *Rx Engine*, the *Timers*, and the *Tx App If*. Consequently, events are merged and forwarded to the *Tx Engine* that processes them to generate the corresponding outgoing packets. Each event will trigger the generation of a new TCP packet. The final event stream is then merged with outstanding acknowledgments [183].

Buffering and Window Management: since TCP is a stream-based protocol, it requires buffering on the receiving and transmitting side. On the receiving side, data is buffered in case the application is not able to immediately consume it. On the sending side, buffering is required for re-transmission in case of packet loss. Thus, when supporting multiple connections, the amount of memory that is needed increases linearly with

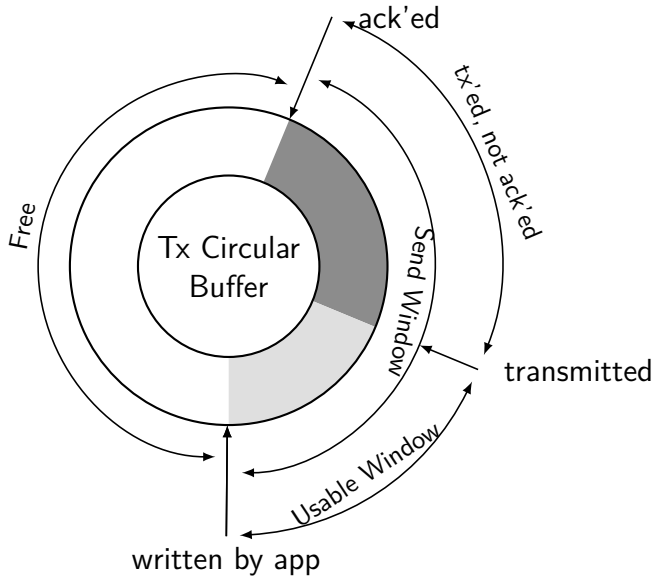


Figure 7.6: TX circular buffer representing the TCP window.

the number of connections. For more than ten concurrent connections, the routing of on-chip memory becomes very complex and using DRAM to store the payloads becomes therefore mandatory. For every connection the memory buffer is logically implemented as a circular buffer which is stored in a fixed and pre-allocated segment within the off-chip memory. Stored in the **Tx and Rx SAR Tables** there are pointers, for instance, *ack'ed*, *transmitted*. These pointers represent the state of the TCP sliding window of each connection at a given time. The information stored in these tables is mandatory to handle the segmentation and reassembly (SAR) of packets as well as maintaining the TCP window. For instance, the *Tx SAR Table* keeps track of three partitions: transmitted but not yet acknowledged, data written by the application to the buffer but not yet transmitted, and finally free space, see Figure 7.6. In addition to these three pointers, the *Tx SAR Table* also stores the *Send Window* which is advertised by the other device and represents the size of its receive buffer. The *Usable Window*, as shown in the figure, can be computed on the fly and is not explicitly stored in the table. Moreover, to support the *Window Scale* TCP option, the *Tx SAR Table* stores the *Window Scale* negotiated when the connection is established. This value defines the size and boundaries of the buffer.

Statistics: this module gathers events for inbound and outbound packets. The values can be read through an AXI4-Lite interface using the DMA subsystem. It provides access to received packets and bytes as well as transmitted packets and bytes and re-transmissions per connection. This element is optional and can be removed at synthesis.

7.6.3 Tx Engine

Each event triggers the generation of a new packet; the packet generation is done by the *Tx Engine*. The source of new packets can be the user application by either initiating a data transmission or by opening a new connection, which triggers a SYN packet. The *Rx Engine* generates events that create ACK packets, including SYN-ACK. The *Timers* module triggers timeout-related events, such as re-transmission, probe packets and FIN packets for teardown. Like the *Rx Engine*, the *Tx Engine* has a FSM to handle each possible event. Contrary to *Rx Engine*, since each event carries the sessionID and event type, the sessionID is known when the event arrives. Consequently, the data-structures are queried immediately getting the necessary metadata to generate the packet. The Destination IP address and TCP ports are queried from the *Session Lookup*. Once the metadata is retrieved, the TCP pseudo header can be built. If the packet has payload, it is fetched from the external memory or directly from the application. Prepending the TCP pseudo header with the payload, the *Tx Checksum* computes the TCP checksum. After that, the checksum is inserted in the TCP header and the pseudo header is removed. Later, the IP header is prepended to the TCP packet. Finally, the packet is forwarded to the *Outbound Packet Handler*.

7.7 Limago Evaluation

7.7.1 Experiments Setup

Limago was designed following a bottom-up approach in order to introduce new functionality in an incremental manner while retaining the ability to easily isolate behavior in the new pieces of hardware. The majority of Limago is programmed in Vivado-HLS, which helped with code productivity as well as maintaining a more flexible design methodology.

The evaluation of Limago covers both functionality and performance. In terms of functionality, first the ARP and ICMP modules were tested using GNU/Linux `arping` and `ping` programs. Then, to test the TOE, we implemented an echo server transmitting the received payload back to the sender. Such a design allows to test both the Rx and Tx Engines. The same echo server is used to verify the correct functionality of the internal elements as well as verifying connectivity.

For the performance evaluation, we use iPerf [93] version 2. We implemented iPerf in hardware, using Vivado-HLS, supporting both client and server modes. As a client, the application actively opens a connection and sends data at the highest possible rate to the server. The transmission duration is specified by a parameter; after termination the

throughput is calculated using the equation $\frac{sent\_bytes}{time}$. As a server, the application waits for a SYN packet to establish a new connection. Once the connection is established, the client starts transmitting data and the application on the FPGA consumes the incoming payload while the TOE acknowledges the received packets. Client and server mode can be used at the same time, for the client mode, the user has to specify the remote host address, remote host port, and the test duration. We have also built a user program on top of the Xilinx DMA driver to interact with the iPerf application deployed on the FPGA.

Limago was tested using two different configurations (Figure 7.7). Scheme 1 corresponds to a standard implementation, each TOE communicates with the corresponding CMAC, and a 100G cable connects both CMACs. In this case the maximum throughput is limited by the Ethernet connection. Scheme 2 removes the Ethernet CMAC and connects the TOE using a 512-bit AXI4-Stream interface clocked at 322 MHz. The idea behind this configuration is to verify the maximum throughput. In the second configuration, we also have tested replacing DDR4 with a 512 KiB URAM — Scheme 2(b). This allows us to verify the physical bounds for each part of the design.

7.7.2 Throughput

To test the throughput of Limago we measured the throughput under two configurations (Figure 7.7) one to test the throughput over a network and another to test the maximum processing rate of Limago when not limited by the network. In this experiment, TOE_0 transmits data to TOE_1, *i.e*, only the memory attached to TOE_0 is involved. The throughput reported measures the complete Ethernet frame, *i.e.*, including the Ethernet, IP and TCP headers as well as the payload. In this experiment, the application

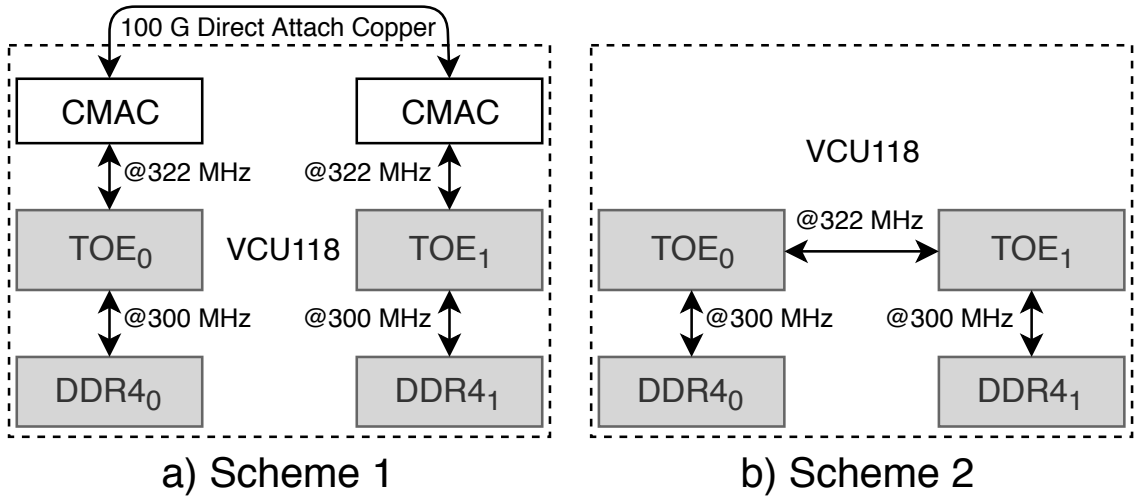


Figure 7.7: Limago interconnection schemes to evaluate potential bottlenecks.

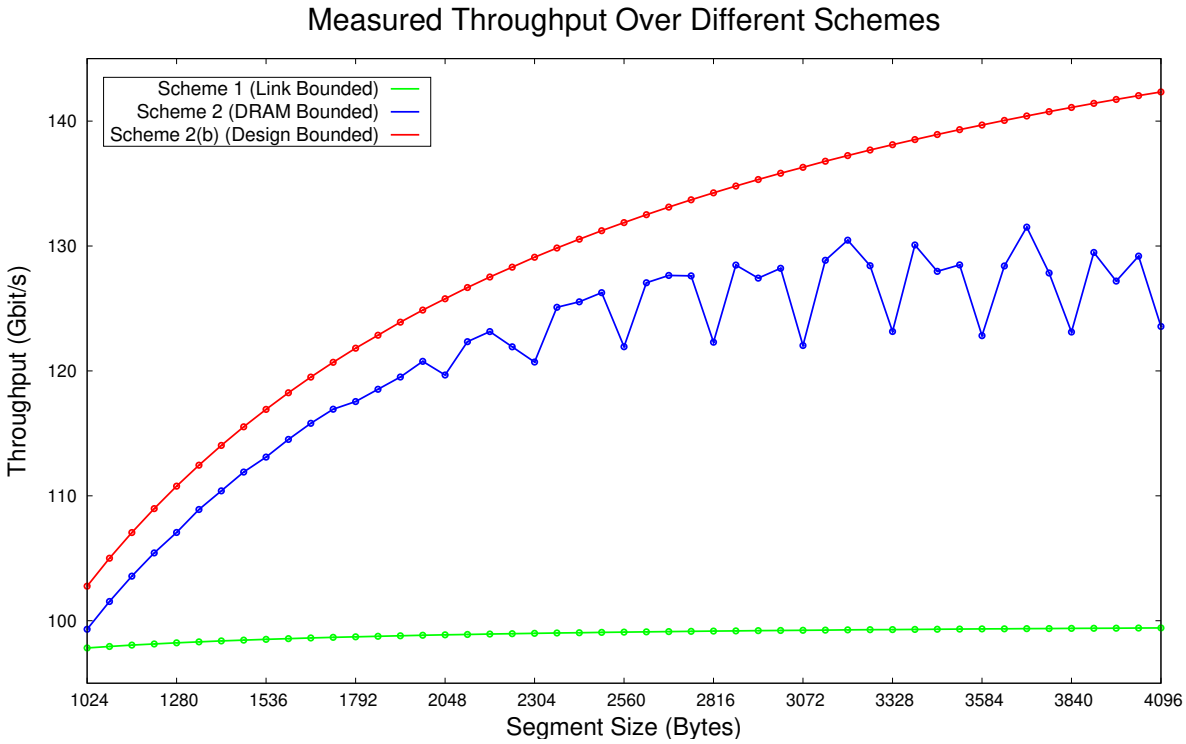


Figure 7.8: Limago performance over different schemes.

transmitted segments ranging between 1024-Byte to 4096-Byte in steps of 64-Byte, using only one connection, each experiment lasted five minutes. Figure 7.8 plots the result of each experiment. For scheme 1, using external DRAM and transmitting packets over the 100 Gbit/s Ethernet link, the throughput is bound by the network — follows the theoretical 100 Gbit/s line-rate as expected. Scheme 2, using DRAM, Limago transmits more than 100 Gbit/s for all cases. However, beyond 2048-Byte segment size, the DRAM bandwidth limits the throughput. Scheme 2(b), using on-chip URAM, looks like a logarithmic function where the throughput increases with an increasing segment size. These experiments show that Limago is able to surpass 100 Gbit/s when it is not bound by the network.

We also have carried out experiments with multiple connections at the same time. For those experiments we have used two servers and a Huawei cloudEngine 8800 switch. The specifications of the severs are as follows: both servers run on a 4.14.7-gentooHPC OS and use a Mellanox MT27800 ConnectX-5 100 Gbit/s NIC; server A has an Intel Xeon CPU E5-2630 v4 at 2.20 GHz and 128 GB of RAM memory, whereas, server B has an Intel Xeon Gold 6126 CPU at 2.60 GHz and 192 GB of RAM memory. All offloading capabilities have been enabled in both machines, using `ethtool`. We use iPerf (version 2) to test the performance, this time the servers work as a client, which means they send the data. Three different scenarios have been evaluated, each server individually and

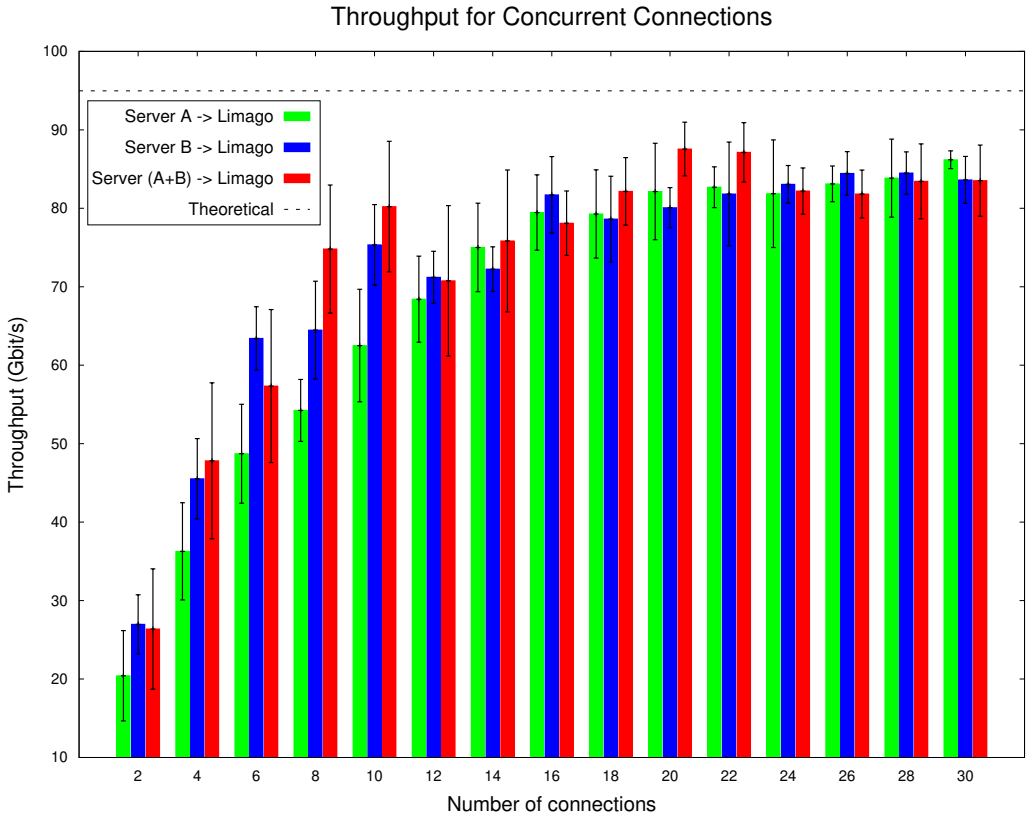


Figure 7.9: Throughput for concurrent connections, mean and standard deviation.

both servers combined. For both servers combined, each one contributes with half of the connections. The number of concurrent connections range from two to thirty in steps of two, each test lasted five minutes and was repeated five times, we set the packet size to 1460-Byte, which is the maximum that guarantee no IP fragmentation. Figure 7.9 shows the results — which are measured at the application level — the mean and standard deviation are plotted, as well as the theoretical maximum. In general, the performance increases with a higher number of concurrent connections, until it is stable. With regard to both servers sending data simultaneously, a better performance is not observed, from this we notice that the switch could be the bottleneck. Further experiments are necessary to confirm this.

7.7.3 Latency Measurement

We have performed latency measurements in Limago using the iperf2 application, in particular, the setup time and the acknowledgment to the first segment time as depict in Figure 7.10, in such figure the latencies are labeled as Δt_{setup} and $\Delta t_{segment}$. We set an observation point next to the TCP module, both ingress and egress. Therefore, we are able to capture the exact time at what the events happen. We have carried out the

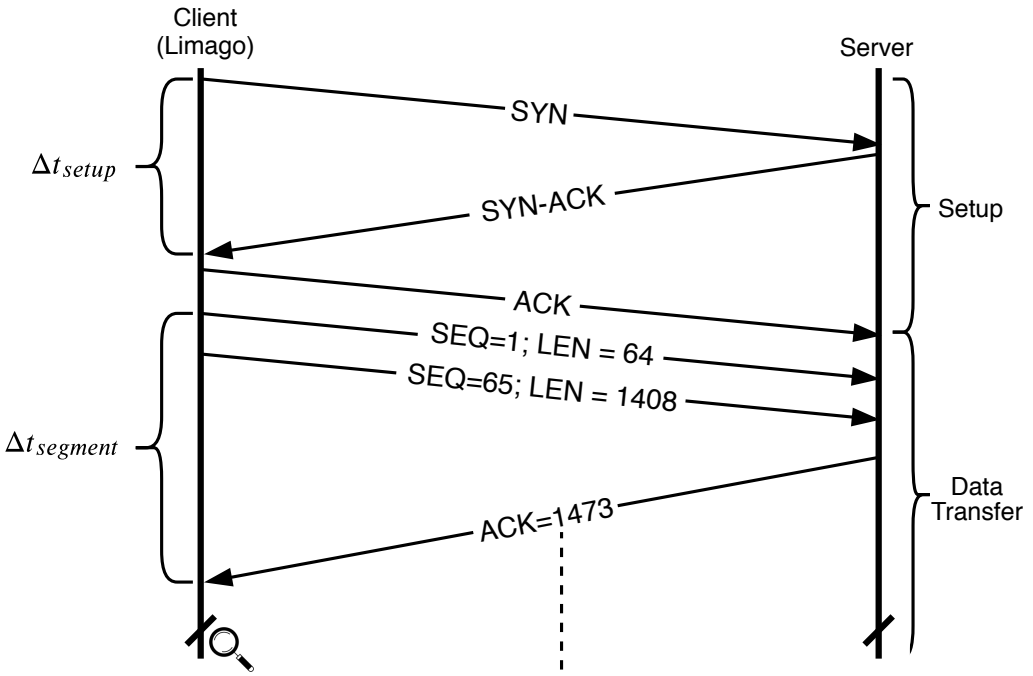


Figure 7.10: Diagram of events where latency was measured in Limago.

latency measurement under two scenarios, Limago → Limago and Limago → server A and server B. For both servers, we used the vanilla configurations, and left the operating system select which core to use in order to run the application. The same experiment was repeated using the Huawei Cloud Engine 8800 switch in between. Each experiment was repeated five times, therefore we plot the arithmetic mean and standard deviation in Figure 7.11, the left side shows the Δt_{setup} time. On the other hand, the right side shows the $\Delta t_{segment}$ time. Additionally, we have evaluated the effect of offloading part of the TCP stack to the NIC as well. In both sides the gray box shows the measurement when the end-points were connected using the switch. The figure shows that Limago to Limago is at least 15 times faster than when there is software involved in the experiment — $\Delta t_{setup} = (1213.27 \pm 23.53)$ nanoseconds and $\Delta t_{segment} = (1424.9 \pm 7.08)$ nanoseconds. Regarding the offloading, we witness a latency increase when the offloading is enabled, this may be caused of the process being allocated in a different NUMA node, however we do not dive deep in this matter. What is more, Limago presents a unique opportunity to evaluate the software latency with both high accuracy and high precision. This kind of experiments are left to future work due to the huge amount of variables to evaluate in the software side, for instance evaluating the effect of allocating the application in a different NUMA node that the one connected to the NIC.

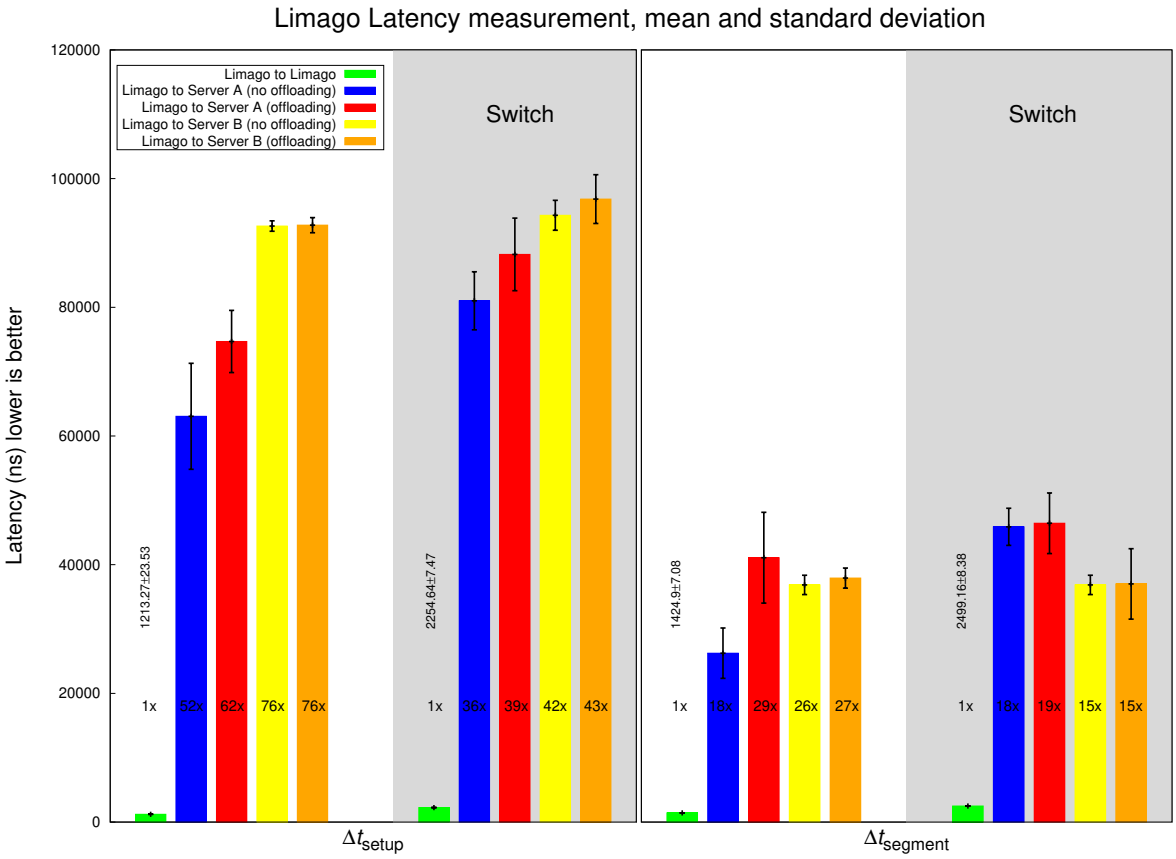


Figure 7.11: Latency measurement on Limago for Δt_{setup} and $\Delta t_{segment}$ time under different connection schemes.

7.7.4 Resource Usage and Code Complexity

Limago has been implemented using Vivado and Vivado HLS 2018.2. The prototype uses a VCU118 board with a Virtex Ultrascale+ FPGA. Figure 7.12 shows the BRAM usage of the TOE for a wide variety of number of connections at a specific Window Scale. The LUTs and Flip-Flop cost are omitted due to small variation between the different scenarios — ranging between 36 K to 41 K. As explained earlier and confirmed by the actual BRAM usage, the data structures in our implementation scale linearly with the number of supported connections.

The resource usage of Limago for 10,000 connections and no Window Scale is listed in Table 7.1. The overall LUT usage is at 10% whereby 3.1% is used by the TOE. The TOE is also 1.5% of the available Flip-Flops which is around 20% of the total usage. But at the same time 90% of the logic resources are still available and can be used to deploy an application on the FPGA. BRAM capacity is a scarcer resource, the TOE uses almost 12% of them, overall around 80% of BRAM and 100% URAM capacity is still available for further use. The table also shows the resource summary for the 10 Gbit/s starting point

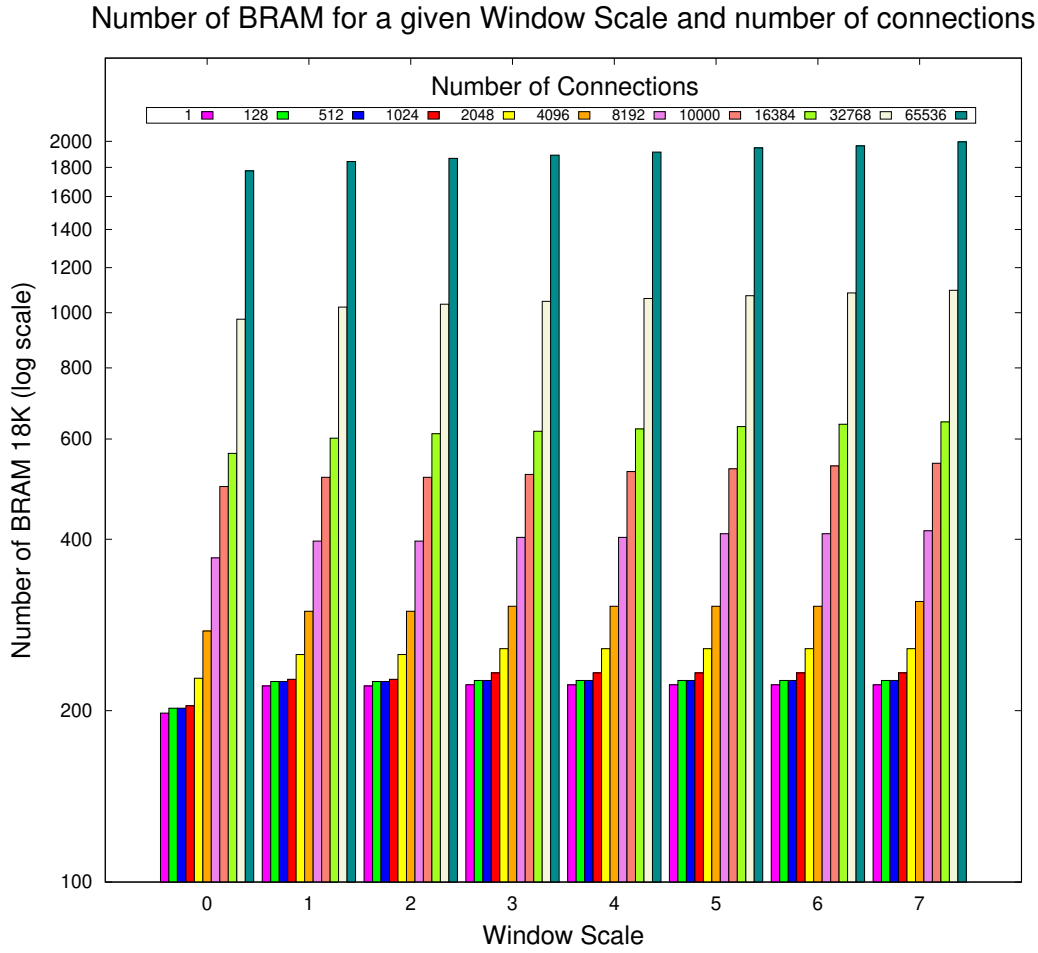


Figure 7.12: TOE BRAM18 usage for different window scale and maximum number of connections.

implementation, for the same FPGA, the resources of the TOE increased by a factor of 1.2 to 2.1. The overall logic resources increased by a factor of two and the BRAM usage by 20%. Particularly noteworthy, the tenfold bandwidth increase, at worse, only requires twice as much resources. This fact is also in part because of Limago targets a FPGA two generation ahead, with a technology node of 16 nm compare against the 28 nm of the starting point implementation.

Limago has ten core modules, seven of them are written in HLS. Apart from the checksum, the other two HDL modules are straightforward, however determinism is needed. We used cloc [184] to count the lines of code (no headers), the HLS part is 7,456 lines; whereas the HDL is 1,482 lines.

Element	LUT		FF		BRAM	
10 Gbit/s Implementation						
TOE	15,415	1.3%	16,616	0.7%	186.5	8.7%
SmartCAM	2,201	0.2%	1,772	0.1%	57.5	2.7%
Total	**77,393**	**6.6%**	**85,306**	**3.6%**	**369**	**17.1%**
100 Gbit/s Implementation						
Memory	17,423	1.5%	25,995	1.1%	41.5	1.9%
CMAC	14,614	1.2%	39,550	1.7%	26.5	1.2%
ARP	1,260	0.1%	3,193	0.1%	1.5	0.1%
ICMP	2,056	0.2%	5,561	0.2%	0.0	0.0%
Inbound	1,816	0.2%	6,293	0.3%	8.5	0.4%
Outbound	2,680	0.2%	9,324	0.4%	34	1.6%
CuckooCAM	2,095	0.2%	1,392	0.1%	36	1.7%
TOE	36,469	3.1%	36,229	1.5%	247.5	11.5%
Total	**119,844**	**10.1%**	**178,339**	**7.5%**	**441.5**	**20.4%**
Comparison						
Increase	**154.85 %**		**209.06 %**		**119.65 %**	

Table 7.1: Full design resource usage on the VCU118 for 10,000 connections. The original implementation and Limago resources are displayed as well as a comparison.

7.8 Conclusions

Limago [185] is an open-source [186] 100 Gbit/s TCP/IP stack that can be implemented on FPGA to enable research and development in programmable NICs and in-network computing. Starting from a pre-existing stack operating at 10 Gbit/s, and designed to support sufficient number of connections in order to operate in a data center, Limago provides a tenfold increase in bandwidth at the cost of a mere 20% increase in BRAM usage, without jeopardizing the ability to support multiple connections of the original design, and maintaining the same design methodology based on Vivado-HLS. The current prototype has been implemented and successfully tested on Xilinx VCU118 and Alveo U200 boards. Limago paves the way to reach a better utilization in heterogeneous infrastructures such as [22, 23]. Finally, Limago can achieve more than 100 Gbit/s, however it is network bounded.

Future work includes further optimizations of the stack to, for instance, enable reordering of out-of-order packets and additional TCP features taking advantage of the increasing availability of High Bandwidth Memory (HBM) in the latest FPGAs. This feature will improve the throughput when packet loss occurs as well as support application level processing [187, 188]. Needless to say, that the price to pay for these kind of features is an increase in the BRAM consumption.

Part IV

Conclusions

CHAPTER 8

CONCLUSIONS

This chapter aims at summarizing and highlighting the main contributions of this Thesis, which have been already described in their respective chapter. Section 8.1 presents the main contribution of each chapter as well as the publications derived of the research. In Section 8.2, we discuss the lessons learned in this Thesis as well as providing suggestions on whether it makes sense to use HLS for network tasks. Finally, in Section 8.3 we outline the future directions, based on the result of this Thesis and the current research trends.

8.1 Contributions

In Chapter 4 (Active Monitoring) we have evaluated active monitoring probes using the packet-train technique in both FPGA-based and Software-based solutions. We were able to quantize the accuracy and precision of both approaches. The experimental results demonstrate that, not only the FPGA-based solution are significantly better than the Software-based counterpart, but also, shorter trains provide the same accuracy in the measure. Therefore, introducing little interference in the network our solutions are able to obtain meaningful Quality of Service (QoS) parameters. What is more, the scalability of the FPGA-based solution is demonstrated with three implementation targeting 1, 10 and 100 Gbit/s. Such results pinpoint the FPGA solution as the most accurate and cost-effective option to measure actively networks. Noteworthy, we were able to design our proof-of-concepts very quickly using HLS in the non-critical parts. We expect that

the FPGA-based solution would provide the same quality of results for the upcoming 200 and 400 Gbit/s link speeds. The hardware implementation of the 1 and 10 Gbit/s implementations was released as open-source [88]. The contribution of this chapter led to the following publications (chronological order) [66, 89, 90, 189]:

- Mario Ruiz, Javier Ramos, Gustavo Sutter, Jorge E. López de Vergara, Sergio López-Buedo and Javier Aracil. **"Leveraging Open Source Platforms and High-Level Synthesis for the Design of FPGA-Based 10 GbE Active Network Probes"**, in 2015 International Conference on ReConFigurable Computing and FPGAs (ReConFig). IEEE, 2015.

- Mario Ruiz, Javier Ramos, Gustavo Sutter, Jorge E. López de Vergara, Sergio López-Buedo and Javier Aracil. **"Accurate and Affordable Packet-train Testing Systems for Multi-Gb/s Networks"**. *IEEE Communication Magazine*, vol. 54, no. 3, pp. 8087, 2016.

- Mario Ruiz, Javier Ramos, Gustavo Sutter, Jorge E. López de Vergara, Sergio López-Buedo and Cristian Sisterna. **"Harnessing Programmable SoCs to Develop Cost-effective Network Quality Monitoring Devices"**, in 2016 26$^{\text{th}}$ International Conference on Field Programmable Logic and Applications (FPL), IEEE, 2016.

- Jorge E. López de Vergara, Mario Ruiz, Lluís Grifre, Marc Ruiz, Luis Vaquero, José Fernando Zazo, Sergio López-Buedo, Oscar González de Dios and Luis Velasco. **"Demonstration of 100 Gbit/s Active Measurements in Dynamically Provisioned Optical Paths"**, in *the 45th European Conference on Optical Communication (ECOC)*.

In Chapter 5 (Passive Monitoring), we discussed the current software-based passive monitoring probes limitations when monitoring 100 Gbit/s computer networks. Consequently, we proposed two FPGA-based bump-in-the-wire implementations that aim at reducing the traffic load in the traditional probes so as to overcome the current processing limitations. On the one hand, we take advantage of the high degree of parallelism of FPGAs to detect cyphered packet and then capping them section 5.3— we estimate a 60 % traffic reduction with this method. On the other hand, we leveraged the on-chip BRAM memory to implement a BRAM-based shift-register to detect duplicate packets and remove them, the best implementation is able to compare the current packet against a sliding windows of 307,200 elements which in the best case translates into a 37.79 millisecond time window (section 5.4.6). Moreover, in this chapter we show how to use a mix approach of HLS and HDL in order to overcome the limitations of HLS in critical

parts of the design. The contribution of this chapter led to the following publications (chronological order) [133, 190]:

- Mario Ruiz, Gustavo Sutter, Sergio López-Buedo and Jorge E. López de Vergara. **"FPGA-based encrypted network traffic identification at 100 Gbit/s"**, in 2016 International Conference on ReConFigurable Computing and FPGAs (ReConFig). IEEE, 2016.

- Mario Ruiz, Gustavo Sutter, Sergio López-Buedo, José Fernando Zazo and Jorge E. López de Vergara. **"An FPGA-Based Approach for Packet Deduplication in 100 Gigabit-per-Second Networks"**, in 2017 International Conference on ReConFigurable Computing and FPGAs (ReConFig). IEEE, 2017.

In Chapter 6 (Checksum Offloading), we discussed the complexity and computational cost of the one's complement checksum computation. The results show a degradation in the performance of the Linux/GNU TCP/IP stack when the CPU is in charge of computing the one's complement checksum. Hence, for handling 100 Gbit/s TCP/IP stack related tasks in an FPGA we were able to reduce the problem to the addition of 33 × 16-bit words in a period of 3.1 ns. Consequently, we studied different alternatives to tackle such problem. Based on our results, the only option to achieve such a task is to use Carry Save Adder (CSA). Therefore, the best alternative is a low level series of reductions harnessing 7 to 3 CSA (Figure 6.7). The different alternatives studied in this chapter were made open-source [149]. In this case, using HDL is the only solution able to implement efficiently such handcrafted architecture. The contribution of this chapter led to the following publications (chronological order) [148, 191]:

- Mario Ruiz, Tobías Alonso, Gustavo Sutter and Sergio López-Buedo. **"FPGA Efficient Checksum Computation for Multi-Gigabits per Second Networks"**, in *III Jornadas de Computación Empotrada y Reconfigurable (JCER2018)*.

- Gustavo Sutter, Mario Ruiz, Sergio López-Buedo, and Gustavo Alonso. **"FPGA-based TCP/IP Checksum Offloading Engine for 100 Gbps Networks"**, in 2018 International Conference on ReConFigurable Computing and FPGAs (ReConFig). IEEE, 2018.

In Chapter 7 (Reliable Data Transmission) we discussed the new paradigm of FPGA in distributed environment (network-attached) and in network data processing. Consequently, the need for a scalable and high-speed TCP/IP stack has arisen. At the same time, a flexible and high productive design methodology is sought, consequently our option is to use HLS. Therefore, we present Limago the first complete open-source TCP/IP stack

implementation at 100 GbE. In this context, we introduced the challenges to tenfold the starting point implementation as well as introducing improvements to achieve 100 GbE. The results show an unprecedented throughput as well as very low latency while using a small portion of the resources. Noteworthy, more than 85 % of the line of codes were written in C/C++. What is more, Limago paves the way for more efficient detached FPGAs and shows the benefits of using HLS for implementing such a complex protocol at high-speed. As a result, Limago has been made open-source [186, 192]. The contribution of this chapter led to the following publication [185]:

- Mario Ruiz, David Sidler, Gustavo Sutter, Gustavo Alonso and Sergio López-Buedo. **"Limago: an FPGA-based Open-source 100 GbE TCP/IP Stack"**, in 2019 29th International Conference on Field Programmable Logic and Applications (FPL), IEEE, 2019.

The research of chapters 6 and 7 is a collaboration with the Systems Group of the Swiss Federal Institute of Technology in Zürich (ETHz), and part of the work was carried out during Mario Ruiz and Gustavo Sutter stay at such group during the first half of 2018. Additionally, Limago is being use as the underlying infrastructure of distributed Machine Learning inference using Convolutional Neural Networks. One such example of this was the demonstration of Xilinx at the 29th Field Programmable Logic and Applications conference.

Finally, the use of HLS has been paramount in order to design some of the functionality of the different implementations detailed through this thesis. Especially, Limago is mostly written in C++ and translated into hardware using Vivado-HLS. Finishing such a work would not have been possible in the span of this Thesis if HDL had been used. Such a result validates the use of HLS for network tasks and packet processing, even for a very complex one as TCP/IP. Nonetheless, to tweak critical parts we always can use HDL to implement the most efficient solution and integrate it afterwards in HLS, which currently is even simpler using the RTL Blackbox.

8.2 Discussion

One of the main contribution of this Thesis is the use of HLS in order to implement networking task. In this section, we shed light on the lessons learned on how to identify whether it makes sense to use HLS for a given network task. These rules are extracted from the experience gained in this Thesis and are empirical observations. What is more, we also observed that the Quality of Result (QoR) of Vivado-HLS has highly

improved since 2015. Therefore, we expect that some of today's limitations are tomorrow's strengths.

In terms of packet processing, HLS has demonstrated to handle such a task adequately achieving: i) initiation interval of one clock cycle, ii) an acceptable latency and iii) little resource usage. However, one drawback is the implementation of FIFOs, especially with 512-bit interfaces, where there is little control on the underlying BRAM configuration, and, most of the time the design ends up with BRAM misuse. In particular, in this Thesis we have used HLS for parsing packets and extracting fields from the headers in: the Inbound Packet Handler and Loopback modules of the active probes (section 4.9.2) and the Inbound Packet Handler, ARP, ICMP, and TOE modules of Limago (section 7.5). Nonetheless, most of the times is complex to reuse the code. For that reason, this year (2019) Eran *et al.* [20] introduce a HLS library for code reuse in packet processing to gain even more design productivity.

The TOE module (section 7.6) has also demonstrate the benefits of using HLS so as to implement a very complex module, with several iterations between the different internal parts. Moreover, we were able to evaluate several improvements in the design with little changes on the code. Needless to say, an HDL implementation would have had better results, especially for short packets. However, our experimental results show that Limago reaches the theoretical maximum Ethernet throughput for a 1024-Byte segment size, while having a very low-latency. Certainly, the extraordinary QoR and the high design productivity of HLS make Limago the best example of a design methodology of a complex design for networking tasks. Using only HDL in the most critical parts where a high level of determinism is mandatory and harnessing HLS in the remaining parts.

In the following items we summarize when to use HLS according to our experience:

- ✓ Parsing packets (section 4.9.2 and section 7.5).
- ✓ More than two modules with communication between them (section 5.3.2, section 5.4.3 and section 7.6).
- ✓ Unsure about the best architecture, Network Parameter Calculator (section 4.7.1).
- ✓ Not targeting a specific initiation interval neither latency, Network Parameter Calculator (section 4.7.1).
- ✓ Complex algorithms/designs with multiple iterations, TOE (section 7.6).

In the following items we summarize when the HLS version is not the best solution at the moment:

- ✗ Very handcrafted architectures such as the one described in sections 5.3.3, 5.4.6 and 6.6.
- ✗ Specific scheduling of the operations such as the deduplicate pipeline architecture (section 5.4.6).
- ✗ Combinational modules such as the one's complement checksum computation (section 5.3.3).
- ✗ Low level control of the FPGA fabric resources, for instance, controlling individually each BRAM port as needed (section 5.4.6).
- ✗ Full determinism on the logic, variability in the generation as observed in the 1 Gbit/s Packet Generator (Figure 4.5).
- ✗ High arithmetic intensity with data dependencies, counting the number of printable ASCII on a vector (section 5.3.3) or computing the one's complement checksum (section 6.6).

Nonetheless, this latter list is bounded to be reduced on the future due to expected improvements on the QoR of HLS tools.

In the gray areas we suggest to start developing with HLS and to evaluate the most critical parts using the Vivado-HLS reports. Once, the critical part becomes a bottleneck that cannot be addressed using HLS and impedes to reach the necessary performance only then move to an RTL implementation of the critical part.

8.3 Future Work

Through the thesis we have shed light on how to take advantage of the FPGA capabilities and HLS tools to perform a wide variety of network tasks. Ranging from monitoring, both active and passive probes, and reliable data transmission at a very high-speed using the demanding TCP protocol. We expect that more and more researchers will join the FPGA community to take advantage of their capabilities to carry out more network tasks.

Most of the experiment with the passive monitoring probes were carried out using synthetic traces. Therefore, the evaluation in a real environment remains. Specially to validate the traffic reduction when capping cyphered packets. What is more, such evaluation will bring light into the deduplicate implementation to remove switching duplicates. On the other hand, for actives probes, we have demonstrated the good quality of results of the FPGA implementation even for 100 Gbit/s, where there is not software

implementation to do the same. Therefore, implementing such a probe at the upcoming 200 and 400 is a step forward in order to test such equipment, however, it will require some effort in the packet generator due to the change in the data path.

The heterogeneity of the FPGAs opens up new possibilities. For instance, Xilinx has incorporate HBM memory in some Ultrascale+ devices. In this context, using such memory the selective acknowledgment, another TCP option, may be supported. However, the trade-off between BRAM and performance gain has to be studied. More BRAM is indispensable to keep track of the different segments, while more bandwidth to the off-chip memory is mandatory for non-sequential accesses. Furthermore, the latency experiments in section 7.7.3 open the possibility of measuring accurately the effect of the different variables in the GNU/Linux TCP implementation, even evaluating different TCP/IP implementations.

CHAPTER 9

CONCLUSIONES

Este capítulo resumen y resalta las principales contribuciones de esta tesis, las cuales han sido ya descriptas en su respectivo capítulo. La sección 9.1 presenta las principales contribuciones de cada capítulo, así como las contribuciones a la comunidad científica en forma de publicaciones derivadas de la investigación desarrollada. Mientras que en la sección 9.2 discutimos las lecciones aprendidas en esta tesis y también sugerimos en qué caso tiene sentido usar síntesis de alto nivel para tareas de redes. Finalmente, en la sección 9.3 presentamos los trabajos futuros que pueden surgir a partir de los resultados de esta tesis y las tendencias actuales en el área.

9.1 Contribuciones

En el capítulo 4 (*Active Monitoring*) evaluamos la monitorización activa de redes usando la técnica de trenes de paquetes con soluciones basadas tanto *software* como FPGA. Fuimos capaces de cuantizar la exactitud y precisión de ambas alternativas. Los resultados de los experimentos demuestran que, las FPGAs no sólo son mucho mejores que las soluciones *software*, sino que también con pocos paquetes en el tren se consigue la misma exactitud en las medidas. De esta forma, las soluciones con FPGA son capaces de obtener resultados de calidad de servicio significativos con una interferencia muy baja en la red. A demás, la escalabilidad de las implementaciones FPGA ha sido demostrada con tres implementaciones dirigidas a 1, 10 y 100 Gbit/s. Los resultados obtenidos en esta tesis demuestran que la opción FPGA es la solución más exacta y rentable para realizar

medidas activas en las redes de ordenadores de muy alta velocidad. Notablemente fuimos capaces de diseñar las pruebas de concepto de forma muy rápida usando síntesis de alto nivel en las partes menos críticas del diseño. Vaticinamos que la implementación FPGA para las venideras velocidades de 200 y 400 Gbit/s proveerá la misma calidad de resultados. Las implementaciones FPGA de 1 y 10 Gbit/s se liberaron como código abierto [88]. Las contribuciones de este capítulo resultaron en los siguientes artículos de divulgación científica (ordenados cronológicamente) [66, 89, 90, 189]:

- Mario Ruiz, Javier Ramos, Gustavo Sutter, Jorge E. López de Vergara, Sergio López-Buedo and Javier Aracil. **"Leveraging Open Source Platforms and High-Level Synthesis for the Design of FPGA-Based 10 GbE Active Network Probes"**, in 2015 International Conference on ReConFigurable Computing and FPGAs (ReConFig). IEEE, 2015.

- Mario Ruiz, Javier Ramos, Gustavo Sutter, Jorge E. López de Vergara, Sergio López-Buedo and Javier Aracil. **"Accurate and Affordable Packet-train Testing Systems for Multi-Gb/s Networks"**. *IEEE Communication Magazine*, vol. 54, no. 3, pp. 8087, 2016.

- Mario Ruiz, Javier Ramos, Gustavo Sutter, Jorge E. López de Vergara, Sergio López-Buedo and Cristian Sisterna. **"Harnessing Programmable SoCs to Develop Cost-effective Network Quality Monitoring Devices"**, in 2016 26th International Conference on Field Programmable Logic and Applications (FPL), IEEE, 2016.

- Jorge E. López de Vergara, Mario Ruiz, Lluís Grifre, Marc Ruiz, Luis Vaquero, José Fernando Zazo, Sergio López-Buedo, Oscar González de Dios and Luis Velasco. **"Demonstration of 100 Gbit/s Active Measurements in Dynamically Provisioned Optical Paths"**, in *the 45th European Conference on Optical Communication (ECOC)*.

En el capítulo 5 (*Passive Monitoring*), discutimos las limitaciones actuales de las sondas de monitorización activas basadas en *software* cuando se requiere monitorizar redes de 100 Gbit/s. Es por este motivo que proponemos utilizar soluciones basadas en FPGA que se conectan antes de la solución *software* y que tienen como objetivo reducir el tráfico que llega a la sonda, de esta forma se puede superar las limitaciones actuales. Por otro lado, aprovechamos del alto nivel de paralelismo presente en las FPGAs para detectar paquetes cifrados y luego recortarlos, para sólo quedarnos con octetos más significativos sección 5.3 — con este método estimamos una reducción del 60 % del tráfico. Así mismo, aprovechamos las memorias internas de las FPGAs (BRAM)

para implementar un registro de desplazamiento mejorado, para detectar paquetes duplicados y luego eliminarlos. La mejor implementación es capaz de almacenar 307.200 elementos, lo que se traduce en una ventana temporal de 37,79 ms de tráfico de red en el mejor de los casos (sección 5.4.6). En este capítulo también mostramos cómo usar un enfoque mixto entre síntesis de alto nivel y lenguajes de descripción de *hardware* para superar las limitaciones actuales de la síntesis de alto nivel. Las contribuciones de este capítulo resultaron en los siguientes artículos de divulgación científica (ordenados cronológicamente) [133, 190]:

- Mario Ruiz, Gustavo Sutter, Sergio López-Buedo and Jorge E. López de Vergara. **"FPGA-based encrypted network traffic identification at 100 Gbit/s"**, in 2016 International Conference on ReConFigurable Computing and FPGAs (ReConFig). IEEE, 2016.

- Mario Ruiz, Gustavo Sutter, Sergio López-Buedo, José Fernando Zazo and Jorge E. López de Vergara. **"An FPGA-Based Approach for Packet Deduplication in 100 Gigabit-per-Second Networks"**, in 2017 International Conference on ReConFigurable Computing and FPGAs (ReConFig). IEEE, 2017.

En el capítulo 6 (*Checksum Offloading*), discutimos la complejidad y el costo computacional de la suma de comprobación en complemento a uno. El resultado de los experimentos muestra una degradación en el desempeño del protocolo TCP/IP cuando la CPU está encargada de realizar el computo de la suma de comprobación en complemento a uno. Por lo tanto, para manejar tareas relacionas con el protocolo TCP/IP a 100 Gbit/s fuimos capaces de reducir el problema a la suma de 33 palabras de 16-bit cada una en un tiempo máximo de 3.1 nanosegundos. Consiguientemente, estudiamos diferentes alternativas para abordar dicho problema. La mejor alternativa aprovecha una serie de reducciones de sumadores que evitan el acarreo (CSA por sus siglas en inglés), en particular un CSA de 7-bit a 3-bit (figura 6.7). Las diferentes alternativas propuestas en este capítulo se han liberado como código abierto [149]. En este caso, la única solución viable es usar lenguajes de descripción de *hardware* debido a todas las optimizaciones de bajo nivel necesarias. Las contribuciones de este capítulo resultaron en los siguientes artículos de divulgación científica (ordenados cronológicamente) [148, 191]:

- Mario Ruiz, Tobías Alonso, Gustavo Sutter and Sergio López-Buedo. **"FPGA Efficient Checksum Computation for Multi-Gigabits per Second Networks"**, in *III Jornadas de Computación Empotrada y Reconfigurable (JCER2018)*.

- Gustavo Sutter, Mario Ruiz, Sergio López-Buedo, and Gustavo Alonso. **"FPGA-based TCP/IP Checksum Offloading Engine for 100 Gbps Networks"**, in

2018 International Conference on ReConFigurable Computing and FPGAs (ReConFig). IEEE, 2018.

En el capítulo 7 (*Reliable Data Transmission*), discutimos la nueva función de las FPGAs en entornos distribuidos, así como la necesidad de protocolos de comunicación que se adapten a la necesidad de las aplicaciones. De esta forma, la necesidad de una implementación FPGA del protocolo TCP/IP a muy alta velocidad se hizo latente. Al mismo tiempo, se busca una metodología de diseño flexible y de alta productividad, por este motivo usamos síntesis de alto nivel. Es por ello que presentamos Limago la primera implementación completa de código abierto del protocolo TCP/IP en una FPGA a 100 Gigabit Ethernet. En este sentido, presentamos los desafíos que tienen que ser abordados para lograr un ancho de banda diez veces mayor a su predecesor, pero también discutimos algunas mejoras necesarias a estas velocidades. Los resultados muestran un ancho de banda sin precedentes y una latencia muy baja sólo utilizando una porción pequeña de los recursos disponibles. Notablemente, más del 85 % del código fuente de Limago está escrito en C/C++. Limago allana el camino para comunicaciones más eficientes y con menos latencia y al mismo tiempo muestra los beneficios de la síntesis de alto nivel en un protocolo de tan alta complejidad. Como resultado Limago se ha hecho accesible como código libre [186, 192]. La contribución de este capítulo resultó en el siguiente artículo de divulgación científica [185]:

- Mario Ruiz, David Sidler, Gustavo Sutter, Gustavo Alonso and Sergio López-Buedo. **"Limago: an FPGA-based Open-source 100 GbE TCP/IP Stack"**, in 2019 29th International Conference on Field Programmable Logic and Applications (FPL), IEEE, 2019.

Parte de la investigación llevada a cabo en los capítulos 6 y 7 es una colaboración con el grupo de sistemas de Universidad ETH Zürich y parte del trabajo fue llevado a cabo durante la estancia de Mario Ruiz y Gustavo Sutter en dicho grupo en la primera mitad del año 2018. Adicionalmente, Limago está siendo usado como la infraestructura de comunicación en un sistema de inferencia distribuida con redes neuronales convolucionales. Un ejemplo de esto es la prueba de concepto llevada a cabo por Xilinx en la conferencia *29th International Conference on Field Programmable Logic and Applications*.

Finalmente, el uso de síntesis de alto nivel ha sido primordial en el desarrollo de alguna de las funcionalidades de las diferentes implementaciones llevadas a cabo durante esta tesis. Especialmente en Limago, que está escrito mayoritariamente en C++ y traducido a *hardware* usando Vivado-HLS. Terminar Limago no habría sido posible en el lapso de esta tesis si se hubiera utilizado lenguajes de descripción de *hardware*. Los resultados obtenidos con Limago validan el uso de síntesis de alto nivel

para procesamiento de paquetes de red, incluso para tareas tan complejas como el protocolo TCP/IP. No obstante, para ajustar las partes críticas siempre podemos utilizar lenguajes de descripción de *hardware* para obtener la solución más eficiente e integrarla posteriormente con la síntesis de alto nivel, este paso es relativamente sencillo utilizando RTL Blackbox.

9.2 Discusión

Una de las principales contribuciones de esta tesis es el uso de síntesis de alto nivel para solucionar tareas de redes. En esta sección arrojamos luz sobre las lecciones aprendidas sobre cómo identificar en qué casos tiene sentido usar síntesis de alto nivel para solucionar una tarea de red determinada. Estas reglas son observaciones empíricas que se extraen de la experiencia adquirida en esta tesis. Es más, también observamos que la calidad de resultado de Vivado-HLS ha mejorado mucho desde 2015. Por lo tanto, esperamos que algunas de las limitaciones de hoy sean las fortalezas del mañana.

Con respecto al procesado de paquetes, la síntesis de alto nivel ha demostrado ser capaz de manejar dicha tarea adecuadamente consiguiendo: i) intervalo de iniciación de un ciclo de reloj, ii) una latencia aceptable y iii) bajo uso de recursos. Sin embargo, una de las limitaciones actuales se encuentra al implementar FIFOs, especialmente cuando el ancho del bus es de 512-bit, en este caso hay muy poco control sobre la configuración de la BRAM y muchas veces se desperdicia parte de su almacenamiento. En particular, en esta tesis usamos síntesis de alto nivel para diseccionar y extraer campos de la cabecera de los paquetes, por ejemplo, en los módulos *Inbound Packet Handler* y *Loopback* de las sondas activas (sección 4.9.2) y en los módulos *Inbound Packet Handler, ARP, ICMP* y *TOE* de Limago (sección 7.5). No obstante, muchas veces es complejo reutilizar código. Por este motivo, este año (2019) Eran *et al.* [20] presentó una biblioteca para síntesis de alto nivel para aumentar la reusabilidad del código para procesado de paquetes con el objetivo de obtener mayor productividad en la etapa de diseño.

El módulo TOE (sección 7.6) también ha demostrado los beneficios de utilizar síntesis de alto nivel para implementar un módulo muy complejo, con varias iteraciones entre las diferentes partes internas. Además, pudimos evaluar varias mejoras en el diseño con pocos cambios en el código fuente. Demás está decir que una implementación usando lenguajes de descripción de *hardware* daría mejores resultados, especialmente para paquetes cortos. Sin embargo, los resultados experimentales muestran que Limago alcanza el rendimiento máximo teórico de Ethernet para un tamaño de segmento de 1024-Bytes, mientras que tiene una latencia muy baja. Ciertamente, el extraordinario rendimiento y la alta productividad de diseño de síntesis de alto nivel hacen de Limago

el mejor ejemplo de una metodología de diseño de cómo implementar una tarea de red extremadamente compleja. Utilizando sólo lenguajes de descripción de *hardware* en las partes más críticas donde es obligatorio un alto nivel de determinismo y aprovechando síntesis de alto nivel en las partes restantes.

En la siguiente lista resumimos cuando utilizar síntesis de alto nivel de acuerdo a la experiencia obtenida en esta tesis:

- ✓ Diseccionar paquetes (sección 4.9.2 y sección 7.5).
- ✓ Más de dos módulos que se comunican entre ellos (sección 5.3.2, sección 5.4.3 y sección 7.6).
- ✓ Desconocimiento de la mejor arquitectura, *Network Parameter Calculator* (sección 4.7.1).
- ✓ Sin objetivo fijo de intervalo de iniciación o latencia máxima, *Network Parameter Calculator* (sección 4.7.1).
- ✓ Algoritmos o implementaciones complejas con múltiples iteraciones, TOE (sección 7.6).

En la siguiente lista resumimos cuando la versión actual de la herramienta de síntesis de alto nivel no es la mejor solución al menos por el momento:

- ✗ Arquitecturas muy artesanales como la descrita en las secciones 5.3.3, 5.4.6 y 6.6.
- ✗ Secuencia específica de operaciones, como la arquitectura de *deduplicate pipeline* de la sección 5.4.6.
- ✗ Módulos combinacionales como el de la suma de comprobación de complemento a uno sección 5.3.3.
- ✗ Control de muy bajo nivel de los recursos básicos de la FPGA, por ejemplo, control individual de los puertos de una BRAM según se requiera (sección 5.4.6).
- ✗ Determinismo total en el circuito lógico implementado, con síntesis de alto nivel observamos variabilidad en el *Packet Generator* de 1 Gbit/s (figura 4.5).
- ✗ Alta intensidad aritmética con dependencia de datos, por ejemplo, contar la cantidad de caracteres ASCII imprimibles en un vector (sección 5.3.3) o computar la suma de comprobación de complemento a uno (sección 6.6).

No obstante, esta última lista está destinada a reducirse en el futuro debido a las mejoras esperadas en la calidad de resultados de las herramientas de síntesis de alto nivel.

En las áreas grises sugerimos comenzar a desarrollar con síntesis de alto nivel y evaluar las partes más críticas utilizando los reportes de Vivado-HLS. Una vez que la parte crítica se convierte en un cuello de botella que no puede ser abordado usando síntesis de alto nivel e impide alcanzar el rendimiento necesario sólo entonces sugerimos pasar a una implementación de más bajo nivel de la parte crítica.

9.3 Trabajo futuro

A través de la tesis, hemos arrojado luz sobre cómo aprovechar las capacidades de las FPGAs y las herramientas de síntesis de alto nivel para realizar una amplia variedad de tareas de red. Desde monitorización, sondas activas y pasivas, y transmisión fiable de datos a una velocidad muy alta utilizando el exigente protocolo TCP/IP. Esperamos que más y más investigadores se unan a la comunidad FPGA para aprovechar sus capacidades para llevar a cabo tareas de red.

La mayoría de los experimentos en monitorización pasiva han sido realizados utilizando trazas sintéticas. Es por esto que la evaluación de dichas implementaciones en entornos reales queda pendiente. Especialmente para validar la reducción de tráfico lograda cuando se recortan paquetes cifrados. A demás, esta evaluación también proveerá información importante sobre la implementación para reducir duplicados de *switching*. Por otro lado, en el contexto de sondas activas, hemos demostrado la buena calidad de resultados que se pueden lograr con implementaciones FPGAs incluso para velocidades de enlace de 100 Gbit/s, donde no hay implementación de *software* capaz de proveer dichas medidas. Es por ello, que implementar sondas para las velocidades de 200 y 400 Gbit/s es un paso adelante para comprobar el equipamiento, sin embargo, esto requerirá esfuerzo adicional, sobre todo en el generador de paquetes debido al cambio en el camino de los datos.

La heterogeneidad de las FPGA abrió nuevas posibilidades. Por ejemplo, Xilinx ha incorporado memorias HBM en algunos dispositivos de la familia Ultrascale+. En este contexto, aprovechando dicha memoria se podría implementar asentimiento selectivo, otra opción de TCP. Sin embargo, es necesario hacer una evaluación más profunda para entender mejor el incremento en memoria BRAM, y si esto compensa la ganancia en desempeño. Se necesita utilizar más BRAM para almacenar el estado de los segmentos, mientras que se necesita más ancho de banda a la memoria cuando no haya accesos secuenciales. Por otro lado, los experimentos de latencias llevados a cabo en la sección

7.7.3 abren la posibilidad de medir de forma muy exacta los efectos de las diferentes variables presentes en el manejo del protocolo TCP en GNU/Linux, incluso se podrían evaluar diferentes implementaciones del protocolo.

APPENDIX A

LIST OF PUBLICATIONS

Publications Directly Related to this Thesis

Indexed Journals and Magazines

1. Mario Ruiz, Javier Ramos, Gustavo Sutter, Jorge E. López de Vergara, Sergio López-Buedo and Javier Aracil. March of 2016. **"Accurate and Affordable Packet-Train Testing Systems for Multi-Gigabit-per-Second Networks"**. Published on *IEEE Communication Magazine*. ISSN: 0163-6804.

   Indexed in JCR 2018. Impact Factor: **10.356**. **Q1** in Engineering, Electrical & Electronic (**7 out of 265**) and **Q1** in Telecommunications (**3 out of 88**).

   Indexed in Scimago 2018. SJR: **2.37**. **Q1** in Computer Science: Computer Networks and Communications; Computer Science Applications. **Q1** in Engineering: Electrical and Electronic Engineering.

   This scientific contribution is part of Chapter 4: Active Monitoring.

International Conferences

2. Mario Ruiz, David Sidler, Gustavo Sutter, Gustavo Alonso and Sergio López-Buedo. September of 2019. **"Limago: an FPGA-based Open-source 100 GbE TCP/IP Stack"**. Accepted for its publication on *29th International Conference on Field Programmable Logic and Applications (FPL 2019)*. DOI: 10.1109/FPL.2019.00053.

GGS Class: "**2**"; GSS Rating "**A-**". Indexed in LiveSHINE as class "**A**" (AvgCitations: **10.28**). Indexed in Microsoft Academic as class "**A-**" (AvgCitations: **12.45**).

This scientific contribution is part of Chapter 7: Reliable Data Transmission.

3. Gustavo Sutter, Mario Ruiz, Sergio López-Buedo, and Gustavo Alonso. December of 2018. **"FPGA-based TCP/IP Checksum Offloading Engine for 100 Gbps Networks"**. Published on *2018 International Conference on Reconfigurable Computing and FPGAs (Reconfig 2018)*. ISSN: 2640-0472.

   GGS Class "**Work in progress**". Indexed in LiveSHINE as class "**C**" (AvgCitations: **5.08**). Indexed in Microsoft Academic as class "**C**" (AvgCitations: **4.69**).

   This scientific contribution is part of Chapter 6: Checksum Offloading.

4. Mario Ruiz, Gustavo Sutter, Sergio López-Buedo, José Fernando Zazo and Jorge E. López de Vergara. December of 2017. **"An FPGA-Based Approach for Packet Deduplication in 100 Gigabit-per-Second Networks"**. Published on *2017 International Conference on Reconfigurable Computing and FPGAs (Reconfig 2017)*. ISBN: 978-1-5386-3797-5.

   GGS Class "**Work in progress**". Indexed in LiveSHINE as class "**C**" (AvgCitations: **5.08**). Indexed in Microsoft Academic as class "**C**" (AvgCitations: **4.69**).

   This scientific contribution is part of Chapter 5: Passive Monitoring.

5. Mario Ruiz, Gustavo Sutter, Sergio López-Buedo and Jorge E. López de Vergara. December of 2016. **"FPGA-based Encrypted Network Traffic Identification at 100 Gbit/s"**. Published on *2016 International Conference on Reconfigurable Computing and FPGAs (Reconfig 2016)*. ISBN: 978-1-5090-3707-0.

   GGS Class "**Work in progress**". Indexed in LiveSHINE as class "**C**" (AvgCitations: **5.08**). Indexed in Microsoft Academic as class "**C**" (AvgCitations: **4.69**).

   This scientific contribution is part of Chapter 5: Passive Monitoring.

6. Mario Ruiz, Javier Ramos, Gustavo Sutter, Jorge E. López de Vergara, Sergio López-Buedo and Cristian Sisterna. August of 2015. **"Harnessing Programmable SoCs to Develop Cost-effective Network Quality Monitoring Devices"**. Published on *26th International Conference on Field Programmable Logic and Applications (FPL 2016)*. ISSN: 1946-1488.

   Indexed in Scimago 2018. SJR: 0.22. GGS Class: "**2**"; GSS Rating "**A-**". Indexed in LiveSHINE as class "**A**" (AvgCitations: **10.28**). Indexed in Microsoft Academic as class "**A-**" (AvgCitations: **12.45**).

   This scientific contribution is part of Chapter 4: Active Monitoring.

7. Mario Ruiz, Javier Ramos, Gustavo Sutter, Jorge E. López de Vergara, Sergio López-Buedo and Javier Aracil. December of 2015. **"Leveraging Open Source Platforms and High-Level Synthesis for the Design of FPGA-Based 10 GbE Active Network Probes"**. Published on *2015 International Conference on Reconfigurable Computing and FPGAs (Reconfig 2015)*. ISBN: 978-1-4673-9406-2.

   Indexed in Scimago 2015. SJR: 0.13. GGS Class "**Work in progress**". Indexed in LiveSHINE as class "**C**" (AvgCitations: **5.08**). Indexed in Microsoft Academic as class "**C**" (AvgCitations: **4.69**).

   This scientific contribution is part of Chapter 4: Active Monitoring.

Other communications

8. Jorge E. López de Vergara, Mario Ruiz, Lluís Grifre, Marc Ruiz, Luis Vaquero, José Fernando Zazo, Sergio López-Buedo, Oscar González de Dios and Luis Velasco. September of 2019. **"Demonstration of 100 Gbit/s Active Measurements in Dynamically Provisioned Optical Paths"**. Accepted for its publication on *The 45th European Conference on Optical Communication (ECOC)*.

   Indexed in Scimago 2018. SJR: **0.5**. GGS Class "**Work in progress**". Indexed in Microsoft Academic as class "**B**" (AvgCitations: **5.12**).

   This scientific contribution is part of Chapter 4: Active Monitoring.

9. Mario Ruiz, Tobías Alonso, Gustavo Sutter and Sergio López-Buedo. September of 2018. **"FPGA Efficient Checksum Computation for Multi-Gigabits per Second Networks"**. Published on *III Jornadas de Computación Empotrada y Reconfigurable (JCER2018)*.

   This scientific contribution is part of Chapter 6: Checksum Offloading.

Publications related to extensions / derivations of the contributions of this Thesis

International Conferences

10. Tobías Alonso, Mario Ruiz, Gustavo Sutter, Sergio López-Buedo and Jorge E. López de Vergara. April of 2019. **"Towards 100 GbE FPGA-Based Flow Monitoring"**. Published on *X Southern Conference on Programmable Logic (SPL 2019)*. ISBN: 978-1-7281-1363-0.

This scientific contribution helps understanding how an FPGA can help traditional software tools to overcome their performance limitation at very high-speed links and it can be associated to Chapter 5: Passive Monitoring.

11. Tobías Alonso, Mario Ruiz, Ángel López García-Arias, Gustavo Sutter and Jorge E. López de Vergara. August of 2018. **"Submicrosecond Latency Video Compression in a Low-End FPGA-based System-on-Chip"**. Published on *28^{th} International Conference on Field Programmable Logic and Applications (FPL 2018)*. ISSN: 1946-1488.

    GGS Class: "**2**"; GSS Rating "**A-**". Indexed in LiveSHINE as class "**A**" (AvgCitations: **10.28**). Indexed in Microsoft Academic as class "**A-**" (AvgCitations: **12.45**).

    This scientific contribution helps understanding how an FPGA can reduce significantly the latency of a system.

12. José Fernando Zazo, Sergio López-Buedo, Mario Ruiz and Gustavo Sutter. December of 2017. **"A Single-FPGA Architecture for Detecting Heavy Hitters in 100 Gbit/s Ethernet links"**. Published on *2017 International Conference on Reconfigurable Computing and FPGAs (Reconfig 2017)*. ISBN: 978-1-5386-3797-5.

    GGS Class "**Work in progress**". Indexed in LiveSHINE as class "**C**" (AvgCitations: **5.08**). Indexed in Microsoft Academic as class "**C**" (AvgCitations: **4.69**).

    This scientific contribution helps understanding how an FPGA can help traditional software tools to overcome their performance limitation at very high-speed links and it can be associated to Chapter 5: Passive Monitoring.

Other communications

13. Tobías Alonso, Mario Ruiz, Gustavo Sutter, Cristian Sisterna, Sergio López-Buedo and Jorge E. López de Vergara. September of 2018. **"Monitorización con FPGAs de flujos y sesiones TCP en enlaces de 40 Gbit/s"**. Published on *III Jornadas de Computación Empotrada y Reconfigurable (JCER2018)*.

    This scientific contribution helps understanding how an FPGA can help traditional software tools to overcome their performance limitation at very high-speed links and it can be associated to Chapter 5: Passive Monitoring.

APPENDIX B

RÉSUMÉ

Résumé

Education

- Ph.D. in Computer Science and Telecommunication, Universidad Autónoma de Madrid. 2015-now
- Five-years degree in Electronic Engineering. Universidad Nacional de San Juan, Argentina, 2009-2014

Current Position

- Ph.D. Student in Computer Science and Telecommunication, Universidad Autónoma de Madrid. 2015-now
- Teaching assistant. Departamento de Tecnología Electrónica y de las Comunicaciones. 2015-now

Teaching

- Training Professor: Dissemination through lectures, seminars and courses of electronic design and computing in ElectraTraining a coordinated project from the School of Engineering of the Autonomous University of Madrid. 2016-now.

- Assistant Professor: Assisting students with their assignments, in different subject and degrees at Autonomous University of Madrid **Computer Organization** and **Computer Architecture** (Computer Science Engineering Degree) **Fundamentals of Microprocessors** (Computer Science Engineering and Telecommunication Technology and Service Engineering Degrees)**Embedded Systems** (Telecommunication Engineering Master's Degree. 2015-now.

- Teaching Assistant: Assisting students with their assignments, Analogue Signal Processing in Electronic Engineering Degree of National University of San Juan. 2012-2014.

Other Research Position

- Research Intern, Research Labs, Xilinx Inc. Dublin, Ireland. February to June 2019.

- Academic Guest, Systems Group, Swiss Federal Institute of Technology in Zürich (ETHz) (Zürich, Switzerland). March to July 2018.

Honors and Awards

- Research Staff Training of Autonomous University of Madrid (FPI-UAM Spanish acronym), Autonomous University of Madrid Scholarship oriented to PhD students. 2016-now

- Open Hardware 2015, Selection of 2015 top Entries.

Participation in Research Projects

1. Racing Drones: Creación del primer juego online multijugador virtualizado y real. MINECO/FEDER (RTC-2016-4744-7). Universidad Autónoma de Madrid, 2017-2019.

2. TRÁFICA: Análisis de tráfico para inteligencia operativa. MINECO (TEC2015-69417-C2-1-R). Universidad Autónoma de Madrid, 2016-now.

3. IDEALIST: Industry-Driven Elastic and Adaptive Lambda Infrastructure for Service and Transport Networks. European Union (FP7-317999). Universidad Autónoma de Madrid, 2012-2015.

4. PACKTRACK: Packet Tracking a Alta Velocidad Para Seguimiento De Tráfico en Redes de Datos. Plan Nacional de I+D, MINECO (TEC2012-33754). Universidad Autónoma de Madrid, 2012-2016.

Assistance to courses and seminars

1. Automatic Hierarchical Parallelization of Linear Recurrence. Computing Platforms Seminar Series (COMPASS). ETH Zürich. 6 July 2018.

2. Low-Latency Analytics on Colossal Data Streams with SummaryStore. Computing Platforms Seminar Series (COMPASS). ETH Zürich. 15 June 2018.

3. A Formally Verified NAT. Launch Seminar, ETH Zürich. 1 June 2018.

4. FLIN: Enabling Fairness and Enhancing Performance in Modern NVMe Solid State Drivers. Launch Seminar, ETH Zürich. 25 May 2018.

5. RAPID: In-Memory Analytical Query Processing Engine with Extreme Performance per Watt. Computing Platforms Seminar Series (COMPASS). ETH Zürich. 24 May 2018.

6. Processing-in-Network (PIN): A programmable NIC for data processing offloading. Launch Seminar, ETH Zürich. 18 May 2018.

7. Towards Interactive Data Exploration. Computing Platforms Seminar Series (COMPASS). ETH Zürich. 16 May 2018.

8. To Push or To Pull: On Reducing Communication and Synchronzation in Graph Computations. Launch Seminar, ETH Zürich. 11 May 2018.

9. Modern programming languages and code generation in the Oracle Database. Computing Platforms Seminar Series (COMPASS). ETH Zürich. 9 May 2018.

10. Covariance Matrix Calculation on a Hybrid FPGA/CPU System. Launch Seminar, ETH Zürich. 4 May 2018.

11. Innovative Applications and Technology Pivots A Perfect Storm in Computing. D-INFK Distinguished Colloquium. Prof. Wen-Mei Hwu. ETH Zürich. 30 April 2018.

12. Characterization of genome structural variation and large inversions using high throughput sequencing. Launch Seminar, ETH Zürich. 27 April 2018.

13. Scaling database systems to high-performance computers. Computing Platforms Seminar Series (COMPASS). ETH Zürich. 26 April 2018.

14. The Challenges and Promises of Large-Scale Biological Imaging. Computing Platforms Seminar Series (COMPASS). ETH Zürich. 19 April 2018.

15. Fast and strongly-consistent per-item resilience in key-value stores. Launch Seminar, ETH Zürich. 13 April 2018.

16. Heterogeneous Computing Systems for Datacenter and HPC Application. Computing Platforms Seminar Series (COMPASS). ETH Zürich. 12 April 2018.

17. Logarithmical Hopping Encoding (LHE) . Compresión rápida para aplicaciones cloud gaming. Universidad Autónoma de Madrid, 13 December 2016.

Bibliography

[1] P. J. Denning and T. G. Lewis, "Exponential Laws of Computing Growth," *Communications of the ACM*, 2017, doi: 10.1145/2976758

[2] Cisco. (2019, Feb) Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update, 2017-2022 White Paper. Accessed: 2019-07-19. [Online]. Available: https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/white-paper-c11-738429.html

[3] A. Lord, A. Soppera, and A. Jacquet, "The Impact of Capacity Growth in National Telecommunications Networks," *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 374, no. 2062, 2016, doi: 10.1098/rsta.2014.0431

[4] N. N. Group. (2018, Jan) Nielsen's Law of Internet Bandwidth. Accessed: 2019-07-19. [Online]. Available: https://www.nngroup.com/articles/law-of-bandwidth/

[5] D. E. Taylor, "Survey and Taxonomy of Packet Classification Techniques," *ACM Computing Surveys (CSUR)*, vol. 37, no. 3, pp. 238–275, 2005, doi: 10.1145/1108956.1108958

[6] H. Chen, Y. Chen, and D. H. Summerville, "A Survey on the Application of FPGAs for Network Infrastructure Security," *IEEE Communications Surveys & Tutorials*, vol. 13, no. 4, pp. 541–561, 2010, doi: 10.1109/SURV.2011.072210.00075

[7] Arista. (2018, Dec) Four Key Trends in the Networked Use of FPGAs. Accessed: 2019-08-12. [Online]. Available: https://www.arista.com/assets/data/pdf/Whitepapers/Trends-in-FPGA-WP.pdf

[8] J. Meng, N. Gebara, H.-C. Ng, P. Costa, and W. Luk, "Investigating the Feasibility of FPGA-based Network Switches," in *2019 IEEE 30th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, vol. 2160. IEEE, 2019, pp. 218–226, doi: 10.1109/ASAP.2019.00010

[9] G. Watson, N. McKeown, and M. Casado, "NetFPGA: A Tool for Network Research and Education," in *2nd workshop on Architectural Research using FPGA Platforms (WARFP)*, vol. 3, 2006.

[10] J. W. Lockwood, N. McKeown, G. Watson, G. Gibb, P. Hartke, J. Naous, R. Raghuraman, and J. Luo, "NetFPGA—An Open Platform for Gigabit-Rate Network Switching and Routing," in *2007 IEEE International Conference on Microelectronic Systems Education (MSE'07)*. IEEE, 2007, pp. 160–161, doi: 10.1109/MSE.2007.69

[11] G. Gibb, J. W. Lockwood, J. Naous, P. Hartke, and N. McKeown, "NetFPGA—An Open Platform for Teaching How to Build Gigabit-Rate Network Switches and Routers," *IEEE Transactions on Education*, vol. 51, no. 3, pp. 364–369, 2008, doi: 10.1109/TE.2008.919664

[12] G. Martin and G. Smith, "High-Level Synthesis: Past,Present, and Future," *IEEE Design & Test of Computers*, vol. 26, no. 4, pp. 18–25, 2009, doi: 10.1109/MDT.2009.83

[13] S. Lahti, P. Sjövall, J. Vanne, and T. D. Hämäläinen, "Are We There Yet? A Study on the State of High-Level Synthesis," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 38, no. 5, pp. 898–911, 2018, doi: 10.1109/TCAD.2018.2834439

[14] A. Cornu, S. Derrien, and D. Lavenier, "HLS Tools for FPGA: Faster Development with better Performance," in *International Symposium on Applied Reconfigurable Computing*. Springer, 2011, pp. 67–78, doi: 10.1007/978-3-642-19475-7_8

[15] M. Forconesi, G. Sutter, S. López-Buedo, J. E. López de Vergara, and J. Aracil, "Bridging the Gap between Hardware and Software Open Source Network Developments," *Network, IEEE*, vol. 28, no. 5, pp. 13–19, sep 2014, doi: 10.1109/mnet.2014.6915434

[16] M. Attig and G. Brebner, "400 Gb/s Programmable Packet Parsing on a Single FPGA," in *Proceedings of the 2011 ACM/IEEE Seventh Symposium on Architectures for Networking and Communications Systems*. IEEE Computer Society, 2011, pp. 12–23, doi: 10.1109/ancs.2011.12

[17] J. F. Zazo, S. López-Buedo, G. Sutter, and J. Aracil, "Automated Synthesis of FPGA-based Packet Filters for 100 Gbps Network Monitoring Applications,"

in *2016 International Conference on ReConFigurable Computing and FPGAs (ReConFig)*. IEEE, 2016, pp. 1–6, doi: 10.1109/ReConFig.2016.7857156

[18] H. Wang, R. Soulé, H. T. Dang, K. S. Lee, V. Shrivastav, N. Foster, and H. Weatherspoon, "P4FPGA: A Rapid Prototyping Framework for P4," in *Proceedings of the Symposium on SDN Research*. ACM, 2017, pp. 122–135, doi: 10.1145/3050220.3050234

[19] P. Benácek, V. Pu, and H. Kubátová, "P4-to-VHDL: Automatic Generation of 100 Gbps Packet Parsers," in *2016 IEEE 24th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, May 2016, pp. 148–155, doi: 10.1109/FCCM.2016.46

[20] H. Eran, L. Zeno, Z. István, and M. Silberstein, "Design Patterns for Code Reuse in HLS Packet Processing Pipelines," in *2019 IEEE 27th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. IEEE, 2019, pp. 208–217, doi: 10.1109/FCCM.2019.00036

[21] X. Yang, Z. Sun, J. Li, J. Yan, T. Li, W. Quan, D. Xu, and G. Antichi, "FAST: Enabling Fast Software/Hardware Prototype for Network Experimentation," in *Proceedings of the International Symposium on Quality of Service*. ACM, 2019, p. 32, doi: 10.1145/3326285.3329067

[22] J. Weerasinghe, F. Abel, C. Hagleitner, and A. Herkersdorf, "Enabling FPGAs in Hyperscale Data Centers," in *2015 IEEE 12th Intl Conf on Ubiquitous Intelligence and Computing and 2015 IEEE 12th Intl Conf on Autonomic and Trusted Computing and 2015 IEEE 15th Intl Conf on Scalable Computing and Communications and Its Associated Workshops (UIC-ATC-ScalCom)*. IEEE, 2015, pp. 1078–1086, doi: 10.1109/UIC-ATC-ScalCom-CBDCom-IoP.2015.199

[23] N. Eskandari, N. Tarafdar, D. Ly-Ma, and P. Chow, "A Modular Heterogeneous Stack for Deploying FPGAs and CPUs in the Data Center," in *Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. ACM, 2019, pp. 262–271, doi: 10.1145/3289602.3293909

[24] S.-W. Jun, M. Liu, S. Lee, J. Hicks, J. Ankcorn, M. King, S. Xu *et al.*, "Bluedbm: An Appliance for Big Data Analytics," in *2015 ACM/IEEE 42nd Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2015, pp. 1–13, doi: 10.1145/2749469.2750412

[25] N. Fujita, R. Kobayashi, Y. Yamaguchi, and T. Boku, "Parallel Processing on FPGA Combining Computation and Communication in OpenCL Programming," in

2019 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW). IEEE, 2019, pp. 479–488, doi: 10.1109/IPDPSW.2019.00089

[26] J. Weerasinghe, R. Polig, F. Abel, and C. Hagleitner, "Network-Attached FPGAs for Data Center Applications," in *2016 International Conference on Field-Programmable Technology (FPT)*. IEEE, 2016, pp. 36–43, doi: 0.1109/FPT.2016.7929186

[27] J. Weerasinghe, F. Abel, C. Hagleitner, and A. Herkersdorf, "Disaggregated FPGAs: Network Performance Comparison against Bare-Metal Servers, Virtual Machines and Linux Containers," in *2016 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*. IEEE, 2016, pp. 9–17, doi: 10.1109/CloudCom.2016.0018

[28] F. Abel, J. Weerasinghe, C. Hagleitner, B. Weiss, and S. Paredes, "An FPGA Platform for Hyperscalers," in *2017 IEEE 25th Annual Symposium on High-Performance Interconnects (HOTI)*. IEEE, 2017, pp. 29–32, doi: 10.1109/HOTI.2017.13

[29] A. A. Al-Aghbari and M. E. Elrabaa, "Cloud-Based FPGA Custom Computing Machines for Streaming Applications," *IEEE Access*, vol. 7, pp. 38 009–38 019, 2019, doi: 10.1109/ACCESS.2019.2906910

[30] D. Sidler, G. Alonso, M. Blott, K. Karras, K. Vissers, and R. Carley, "Scalable 10 Gbps TCP/IP Stack Architecture for Reconfigurable Hardware," in *2015 IEEE 23rd Annual International Symposium on Field-Programmable Custom Computing Machines*. IEEE, 2015, pp. 36–43, doi: 10.1109/FCCM.2015.12

[31] Z. István, D. Sidler, and G. Alonso, "Caribou: Intelligent Distributed Storage," *Proceedings of the VLDB Endowment*, vol. 10, no. 11, pp. 1202–1213, 2017, doi: 10.14778/3137628.3137632

[32] G. Antichi, S. Giordano, D. J. Miller, and A. W. Moore, "Enabling Open-Source High Speed Network Monitoring on NetFPGA," in *2012 IEEE Network Operations and Management Symposium*. IEEE, 2012, pp. 1029–1035, doi: 10.1109/NOMS.2012.6212025

[33] G. Antichi, M. Shahbaz, Y. Geng, N. Zilberman, A. Covington, M. Bruyere, N. McKeown, N. Feamster, B. Felderman, M. Blott *et al.*, "OSNT: Open Source Network Tester," *Network, IEEE*, vol. 28, no. 5, pp. 6–12, 2014, doi: 10.1109/mnet.2014.6915433

[34] A. Oeldemann, T. Wild, and A. Herkersdorf, "FlueNT10G: A Programmable FPGA-based Network Tester for Multi-10-Gigabit Ethernet," in *2018 28th International Conference on Field Programmable Logic and Applications (FPL)*. IEEE, 2018, pp. 178–1787, doi: 10.1109/FPL.2018.00037

[35] K. Karras and J. Hrica, "Protocol-Processing System Thrive with Vivado HLS," *Xcell Journal*, vol. Third Quarter 2014, no. 88, pp. 45–51, 2014.

[36] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese *et al.*, "P4: Programming Protocol-Independent Packet Processors," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 3, pp. 87–95, 2014, doi: 10.1145/2656877.2656890

[37] L. Wirbel, "Xilinx SDNet: A New Way to Specify Network Hardware," *The Linley Group, Mountain View, CA, USA, White Paper*, no. 2014, 2014.

[38] A. Yazdinejad, A. Bohlooli, and K. Jamshidi, "P4 to SDNet: Automatic Generation of an Efficient Protocol-Independent Packet Parser on Reconfigurable Hardware," in *2018 8th International Conference on Computer and Knowledge Engineering (ICCKE)*. IEEE, 2018, pp. 159–164, doi: 10.1109/ICCKE.2018.8566590

[39] S. Ibanez, G. Brebner, N. McKeown, and N. Zilberman, "The P4-> NetFPGA Workflow for Line-Rate Packet Processing," in *Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. ACM, 2019, pp. 1–9, doi: 10.1145/3289602.3293924

[40] N. Sultana, S. Galea, D. Greaves, M. Wójcik, J. Shipton, R. Clegg, L. Mai, P. Bressana, R. Soulé, R. Mortier *et al.*, "Emu: Rapid Prototyping of Networking Services," in *2017 {USENIX} Annual Technical Conference ({USENIX}{ATC} 17)*, 2017, pp. 459–471, doi: 10.17863/CAM.13009

[41] N. Tarafdar, N. Eskandari, V. Sharma, C. Lo, and P. Chow, "Galapagos: A Full Stack Approach to FPGA Integration in the Cloud," *IEEE Micro*, vol. 38, no. 6, pp. 18–24, 2018, doi: 10.1109/MM.2018.2877290

[42] G. Moore, "Moores Law," *Electronics Magazine*, vol. 38, no. 8, p. 114, 1965.

[43] P. K. Bondyopadhyay, "Moore's Law Governs the Silicon Revolution," *Proceedings of the IEEE*, vol. 86, no. 1, pp. 78–81, 1998, doi: 10.1109/5.658761

[44] S. E. Thompson and S. Parthasarathy, "Moore's Law: the Future of Si Microelectronics," *Materials today*, vol. 9, no. 6, pp. 20–25, 2006, doi: 10.1016/S1369-7021(06)71539-5

[45] B. Winston, *Media, Technology and Society: A History: From the Telegraph to the Internet*. Routledge, 2002.

[46] J. Ryan, *A History of the Internet and the Digital Future*. Reaktion Books, 2010.

[47] S. Cherry, "Edholm's law of bandwidth," *IEEE spectrum*, vol. 41, no. 7, pp. 58–60, 2004, doi: 10.1109/MSPEC.2004.1309810

[48] J. Wu, Y.-L. Shen, K. Reinhardt, H. Szu, and B. Dong, "A Nanotechnology Enhancement to Moore's Law," *Applied Computational Intelligence and Soft Computing*, vol. 2013, p. 2, 2013, doi: 10.1155/2013/426962

[49] R. H. Dennard, F. H. Gaensslen, V. L. Rideout, E. Bassous, and A. R. LeBlanc, "Design of ion-implanted MOSFET's with very small Physical Dimensions," *IEEE Journal of Solid-State Circuits*, vol. 9, no. 5, pp. 256–268, 1974, doi: 10.1109/JSSC.1974.1050511

[50] M. Bohr, "A 30 year Retrospective on Dennard's MOSFET Scaling Paper," *IEEE Solid-State Circuits Society Newsletter*, vol. 12, no. 1, pp. 11–13, 2007, doi: 10.1109/N-SSC.2007.4785534

[51] J. L. Hennessy and D. A. Patterson, *Computer Architecture: a Quantitative Approach*. Elsevier, 2011.

[52] M. Horowitz, "Computing's Energy Problem (and What We Can Do About It)," in *2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*. IEEE, 2014, pp. 10–14, doi: 10.1109/ISSCC.2014.6757323

[53] S. M. Trimberger, "Three Ages of FPGAs: A Retrospective on the First Thirty Years of FPGA Technology," *Proceedings of the IEEE*, vol. 103, no. 3, pp. 318–331, 2015, doi: 10.1109/jproc.2015.2392104

[54] B. Gaide, D. Gaitonde, C. Ravishankar, and T. Bauer, "Xilinx Adaptive Compute Acceleration Platform: Versal TM Architecture," in *Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. ACM, 2019, pp. 84–93, doi: 10.1145/3289602.3293906

[55] S. Lee, K. Levanti, and H. S. Kim, "Network Monitoring: Present and Future," *Computer Networks*, vol. 65, pp. 84–98, 2014, doi: 10.1016/j.comnet.2014.03.007

[56] J. L. García-Dorado, F. Mata, J. Ramos, P. M. S. del Río, V. Moreno, and J. Aracil, "High-performance Network Traffic Processing Systems Using Commodity Hardware," in *Data traffic monitoring and analysis*. Springer, 2013, pp. 3–27, doi: 10.1007/978-3-642-36784-7_1

[57] V. Moreno, P. M. S. Del Río, J. Ramos, J. L. G. Dorado, I. Gonzalez, F. J. G. Arribas, and J. Aracil, "Packet Storage at Multi-gigabit Rates Using Off-the-Shelf Systems," in *2014 IEEE Intl Conf on High Performance Computing and Communications, 2014 IEEE 6th Intl Symp on Cyberspace Safety and Security, 2014 IEEE 11th Intl Conf on Embedded Software and Syst (HPCC, CSS, ICESS)*. IEEE, 2014, pp. 486–489, doi: 10.1109/HPCC.2014.81

[58] ARM. AMBA AXI and ACE Protocol Specification. Accessed: 2019-08-03. [Online]. Available: http://www.gstitt.ece.ufl.edu/courses/fall15/eel4720_5721/labs/refs/AXI4_specification.pdf

[59] ARM. AMBA 4 AXI4-Stream Protocol. Accessed: 2019-08-03. [Online]. Available: https://static.docs.arm.com/ihi0051/a/IHI0051A_amba4_axi4_stream_v1_0_protocol_spec.pdf

[60] Xilinx Inc. AXI Reference Guide. Accessed: 2019-08-03. [Online]. Available: https://www.xilinx.com/support/documentation/ip_documentation/axi_ref_guide/latest/ug761_axi_reference_guide.pdf

[61] Xilinx Inc., "UltraScale+ Devices Integrated 100G Ethernet Subsystem v2.4," Xilinx Inc., Tech. Rep., 04 2018. [Online]. Available: https://www.xilinx.com/support/documentation/ip_documentation/cmac_usplus/v2_4/pg203-cmac-usplus.pdf

[62] Xilinx Inc., "VCU108 Evaluation Board, User Guide UG1066," Tech. Rep., 07 2016. [Online]. Available: http://www.xilinx.com/support/documentation/boards_and_kits/vcu108/ug1066-vcu108-eval-bd.pdf

[63] Xilinx Inc., "VCU118 Evaluation Board, User Guide UG1224," Xilinx Inc., Tech. Rep., 05 2018. [Online]. Available: https://www.xilinx.com/support/documentation/boards_and_kits/vcu118/ug1224-vcu118-eval-bd.pdf

[64] Body of European Regulators for Electronic Communications, "Monitoring quality of Internet Access Services in the Context of Net Neutrality," Tech. Rep. BoR (14) 117, sep 2014, doi: 10.1007/s10272-015-0524-4

[65] J. Ramos, P. S. del Río, J. Aracil, and J. L. de Vergara, "On the Effect of Concurrent Applications in Bandwidth Measurement Speedometers," *Computer Networks*, vol. 55, no. 6, pp. 1435–1453, 2011, doi: 10.1016/j.comnet.2010.10.022

[66] M. Ruiz, G. Sutter, S. López-Buedo, J. Ramos, J. L. de Vergara, and J. Aracil, "Leveraging Open Source Platforms and High-level Synthesis for the Design of

FPGA-based 10 GbE Active Network Probes," in *2015 International Conference on ReConFigurable Computing and FPGAs (ReConFig)*. IEEE, 2015, pp. 1–6, doi: 10.1109/ReConFig.2015.7393325

[67] M. Blott, J. Ellithorpe, N. McKeown, K. Vissers, and H. Zeng, "FPGA Research Design Platform Fuels Network Advances," *Xilinx Xcell Journal*, vol. 4, no. 73, pp. 24–29, 2010.

[68] Xilinx Inc. (2019, July) Vivado Design Suite User Guide, High-Level Synthesis. [Online]. Available: https://www.xilinx.com/support/documentation/sw_manuals/xilinx2019_1/ug902-vivado-high-level-synthesis.pdf

[69] A. Tockhorn, P. Danielis, and D. Timmermann, "A Configurable FPGA-Based Traffic Generator for High-Performance Tests of Packet Processing Systems," in *6th International Conference on Internet Monitoring and Protection (ICIMP)*, mar 2011, pp. 14–19.

[70] T. Groleat, M. Arzel, S. Vaton, A. Bourge, Y. Le Balch, H. Bougdal, and M. Aranaz Padron, "Flexible, Extensible, Open-source and Affordable FPGA-based Traffic Generator," in *Proceedings of the First Edition Workshop on High Performance and Programmable Networking*. ACM, 2013, pp. 23–30, doi: 10.1145/2465839.2465843

[71] J. F. Zazo, M. Forconesi, S. López-Buedo, G. Sutter, and J. Aracil, "TNT10G: A High-Accuracy 10 GbE Traffic Player and Recorder for Multi-Terabyte Traces," in *ReConFigurable Computing and FPGAs (ReConFig), 2014 International Conference on*. IEEE, 2014, pp. 1–6, doi: 10.1109/reconfig.2014.7032561

[72] Y. Wang, Y. Liu, X. Tao, and Q. He, "An FPGA-Based High-Speed Network Performance Measurement for RFC 2544," *EURASIP Journal on Wireless Communications and Networking*, vol. 2015, no. 1, p. 2, jun 2015, doi: 10.1186/1687-1499-2015-2

[73] S. Bradner and J. McQuaid, "Benchmarking Methodology for Network Interconnect Devices," RFC 2544 (Informational), Internet Engineering Task Force, mar 1999, doi: 10.17487/RFC2544

[74] R. Olsson, "Pktgen the Linux Packet Generator," in *Proceedings of the Linux Symposium, Ottawa, Canada*, vol. 2, Jul. 2005, pp. 11–24.

[75] I. Corporation. Data Plane Development Kit. [Online]. Available: http://dpdk.org/

[76] M. Jinno, H. Takara, B. Kozicki, Y. Tsukishima, Y. Sone, and S. Matsuoka, "Spectrum-efficient and scalable elastic optical path network: architecture, benefits, and enabling technologies," *IEEE communications magazine*, vol. 47, no. 11, pp. 66–73, 2009, doi: 10.1109/MCOM.2009.5307468

[77] O. Gerstel, M. Jinno, A. Lord, and S. B. Yoo, "Elastic optical networking: A new dawn for the optical layer?" *IEEE Communications Magazine*, vol. 50, no. 2, pp. s12–s20, 2012, doi: 10.1109/MCOM.2012.6146481

[78] A. Lord, "Optical Metro Networks in a 5G World," *44th European Conference on Optical Communications (ECOC 2018)*, 2018.

[79] L. Velasco, L. Gifre, J.-L. Izquierdo-Zaragoza, G. Julián-Moreno, and J. L. de Vergara, "CASTOR: An Architecture to bring Cognition to Transport Networks," in *Optical Fiber Communication Conference*. Optical Society of America, 2018, pp. Tu3D–5, doi: 10.1364/OFC.2018.Tu3D.5

[80] C. Dovrolis, P. Ramanathan, and D. Moore, "What do Packet Dispersion Techniques Measure?" in *INFOCOM 2001. Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings.*, vol. 2, jan 2001, pp. 905–914, doi: 10.1109/infcom.2001.916282

[81] A. Johnsson, "On the Comparison of Packet-Pair and Packet-Train Measurements," in *Proc. Swedish National Computer Networking Workshop*, Sep. 2003, pp. 241–250.

[82] B. Ahlgren, M. Björkman, and B. Melander, "Network Probing Using Packet Trains," 1999.

[83] A. Botta, A. Dainotti, and A. Pescape, "Do You Trust your Software-Based Traffic Generator?" *Communications Magazine, IEEE*, vol. 48, no. 9, pp. 158–165, Sep. 2010, doi: 10.1109/MCOM.2010.5560600

[84] V. Moreno, J. Ramos, P. Santiago del Río, J. García-Dorado, F. Gómez-Arribas, and J. Aracil, "Commodity Packet Capture Engines: Tutorial, Cookbook and Applicability," *Communications Surveys Tutorials, IEEE*, vol. 17, no. 3, pp. 1364–1390, aug 2015, doi: 10.1109/COMST.2015.2424887

[85] Intel Corporation. The Pktgen Application. [Online]. Available: https://pktgen-dpdk.readthedocs.io/en/latest/

[86] R. Leira, J. Aracil, J. E. L. de Vergara, P. Roquero, and I. González, "High-Speed Optical Networks Latency Measurements in the Microsecond Timescale with

Software-Based Traffic Injection," *Optical Switching and Networking*, vol. 29, pp. 39–45, 2018, doi: 10.1016/j.osn.2018.03.004

[87] NetFPGA Project: Open Source HW and SW for Rapid Prototyping of Computer Network Devices. [Online]. Available: http://netfpga.org/

[88] Hardware Packet Train. https://github.com/hpcn-uam/hardware_packet_train.

[89] M. Ruiz, J. Ramos, G. Sutter, J. E. L. de Vergara, S. López-Buedo, and J. Aracil, "Accurate and Affordable Packet-train Testing Systems for Multi-gigabit-per-second Networks," *IEEE Communications Magazine*, vol. 54, no. 3, pp. 80–87, 2016, doi: 10.1109/MCOM.2016.7432152

[90] M. Ruiz, J. Ramos, G. Sutter, S. López-Buedo, J. L. de Vergara, and C. Sisterna, "Harnessing Programmable SoCs to Develop Cost-Effective Network Quality Monitoring Devices," in *2016 26th International Conference on Field Programmable Logic and Applications (FPL)*. IEEE, 2016, pp. 1–4, doi: 10.1109/FPL.2016.7577320

[91] W. Wu, P. DeMar, and M. Crawford, "Why Can Some Advanced Ethernet NICs Cause Packet Reordering?" *Communications Letters, IEEE*, vol. 15, no. 2, pp. 253–255, feb 2011, doi: 10.1109/LCOMM.2011.122010.102022

[92] Miercom, "1 GE Edge Switch Study: Cisco Catalyst 2960X-48LPD-L, HP 2920-48G-PoE+, HP 5120-48G-PoE+ EI," Tech. Rep. DR130917L, jul 2014. [Online]. Available: http://miercom.com/pdf/reports/20130917.pdf

[93] A. Tirumala, F. Qin, J. Dugan, J. Ferguson, and K. Gibbs, "iPerf: the TCP/UDP Bandwidth Measurement Tool (2005)," *URL: http://iperf. sourceforge. net*, 2005.

[94] Xilinx Inc., "Virtex UltraScale+ 56G PAM4 VCU129 FPGA Evaluation Kit," Xilinx Inc., Tech. Rep., 08 2019. [Online]. Available: https://www.xilinx.com/products/boards-and-kits/vcu129-pp.html

[95] V. Moreno, J. Ramos, J. L. García-Dorado, I. González, F. J. Gómez-Arribas, and J. Aracil, "Testing the Capacity of Off-the-Shelf Systems to Store 10GbE Traffic," *IEEE Communications Magazine*, vol. 53, no. 9, pp. 118–125, September 2015, doi: 10.1109/MCOM.2015.7263355

[96] G. Julián-Moreno, R. Leira, J. E. L. de Vergara, F. J. Gómez-Arribas, and I. González, "On the Feasibility of 40 Gbps Network Data Capture and Re-

tention with General Purpose Hardware," in *Proceedings of the 33rd Annual ACM Symposium on Applied Computing*. ACM, 2018, pp. 970–978, doi: 10.1145/3167132.3167238

[97] P. Emmerich, M. Pudelko, S. Gallenmüller, and G. Carle, "FlowScope: Efficient Packet Capture and Storage in 100 Gbit/s Networks," in *2017 IFIP Networking Conference (IFIP Networking) and Workshops*. IEEE, 2017, pp. 1–9, doi: 10.23919/IFIPNetworking.2017.8264852

[98] P. Velan, M. Čermák, P. Čeleda, and M. Drašar, "A Survey of Methods for Encrypted Traffic Classification and Analysis," *International Journal of Network Management*, vol. 25, no. 5, pp. 355–374, 2015, doi: 10.1002/nem.1901

[99] M. Finsterbusch, C. Richter, E. Rocha, J.-A. Muller, and K. Hanssgen, "A Survey of Payload-Based Traffic Classification Approaches," *IEEE Communications Surveys & Tutorials*, vol. 16, no. 2, pp. 1135–1156, 2013, doi: 10.1109/SURV.2013.100613.00161

[100] A. Dainotti, A. Pescape, and K. C. Claffy, "Issues and Future Directions in Traffic Classification," *IEEE network*, vol. 26, no. 1, pp. 35–40, 2012, doi: 10.1109/MNET.2012.6135854

[101] P. Dorfinger, G. Panholzer, and W. John, "Entropy Estimation for Real-Time Encrypted Traffic Identification," in *International Workshop on Traffic Monitoring and Analysis*. Springer, 2011, pp. 164–171, doi: 10.1007/978-3-642-20305-3_14

[102] L. Bernaille and R. Teixeira, "Early Recognition of Encrypted Applications," in *Proceedings of the 8th International Conference on Passive and Active Network Measurement*, ser. PAM'07. Springer-Verlag, 2007, pp. 165–175, doi: 10.1007/978-3-540-71617-4_17

[103] M. Forconesi, G. Sutter, S. López-Buedo, and J. Aracil, "Accurate and Flexible flow-based Monitoring for High-Speed Networks," in *2013 23rd International Conference on Field programmable Logic and Applications (FPL)*, Sep 2013, pp. 1–4, doi: 10.1109/FPL.2013.6645557

[104] T. Alonso, M. Ruiz, G. Sutter, S. López-Buedo, and J. E. L. De Vergara, "Towards 100 GbE FPGA-Based Flow Monitoring," in *2019 X Southern Conference on Programmable Logic (SPL)*. IEEE, 2019, pp. 9–16, doi: 10.1109/SPL.2019.8714532

[105] G. Nassopulos, D. Rossi, F. Gringoli, L. Nava, M. Dusi, and P. M. S. del Rio, "Flow Management at Multi-Gbps: Tradeoffs and Lessons Learned," in *International Workshop on Traffic Monitoring and Analysis*. Springer, 2014, pp. 1–14, doi: 10.1007/978-3-642-54999-1_1

[106] G.-L. Sun, Y. Xue, Y. Dong, D. Wang, and C. Li, "An Novel Hybrid Method for Effectively Classifying Encrypted Traffic," in *Global Telecommunications Conference (GLOBECOM 2010), 2010 IEEE*. IEEE, Dec 2010, pp. 1–5, doi: 10.1109/GLOCOM.2010.5683649

[107] S. Ata, G. Hasegawa, Y. Nakahira, and N. Nakamura, "Encrypted-Traffic Discrimination Device and Encrypted-Traffic Discrimination System," apr 2015, uS Patent 9,021,252. [Online]. Available: https://www.google.com/patents/US9021252

[108] R. Alshammari and A. N. Zincir-Heywood, "Can Encrypted Traffic be Identified without Port Numbers, IP Addresses and Payload Inspection?" *Computer networks*, vol. 55, no. 6, pp. 1326–1350, 2011, doi: 10.1016/j.comnet.2010.12.002

[109] M. Lotfollahi, M. J. Siavoshani, R. S. H. Zade, and M. Saberian, "Deep Packet: A Novel Approach for Encrypted Traffic Classification Using Deep Learning," *Soft Computing*, pp. 1–14, 2017, doi: 10.1007/s00500-019-04030-2

[110] V. Uceda, M. Rodríguez, J. Ramos, J. L. García-Dorado, and J. Aracil, "Selective Capping of Packet Payloads at Multi-Gb/s Rates," *IEEE Journal on Selected Areas in Communications*, vol. 34, no. 6, pp. 1807–1818, June 2016, doi: 10.1109/jsac.2016.2559198

[111] V. Uceda, M. Rodríguez, J. Ramos, J. L. García-Dorado, and J. Aracil, "Selective Capping of Packet Payloads for Network Analysis and Management," in *International Workshop on Traffic Monitoring and Analysis*. Springer, 2015, pp. 3–16, doi: 10.1007/978-3-319-17172-2_1

[112] Sandvine, "Global Internet Phenomena Spotlight: Encrypted Internet Traffic," Tech. Rep., 04 2016. [Online]. Available: https://www.sandvine.com/hubfs/downloads/archive/2016-global-internet-phenomena-spotlight-encrypted-internet-traffic.pdf

[113] Karras, Kimon and Hrica, James. (2014) Designing Protocol Processing Systems with Vivado High-Level Synthesis. Xilinx Inc. [Online]. Available: http://www.xilinx.com/support/documentation/application_notes/xapp1209-designing-protocol-processing-systems-hls.pdf

[114] V. Sklyarov, I. Skliarova, A. Sudnitson, and M. Kruus, "FPGA-Based Time and Cost Effective Hamming weight Comparators for Binary Vectors," in *EUROCON 2015 - International Conference on Computer as a Tool (EUROCON), IEEE*, Sept 2015, pp. 1–6, doi: 10.1109/EUROCON.2015.7313700

[115] Xilinx Inc., "UltraScale Architecture Integrated Block for 100G Ethernet v1.10," Tech. Rep., June 2016. [Online]. Available: http://www.xilinx.com/support/documentation/ip_documentation/cmac/v1_10/pg165-cmac.pdf

[116] A. Anand, A. Gupta, A. Akella, S. Seshan, and S. Shenker, "Packet Caches on Routers: The Implications of Universal Redundant Traffic Elimination," in *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 4. ACM, 2008, pp. 219–230, doi: 10.1145/1402958.1402984

[117] I. Ucar, D. Morato, E. Magana, and M. Izal, "Duplicate Detection Methodology for IP Network Traffic Analysis," in *Measurements and Networking Proceedings (M&N), 2013 IEEE International Workshop on*. IEEE, 2013, pp. 161–166, doi: 10.1109/iwmn.2013.6663796

[118] J. Cabal, P. Benáček, L. Kekely, M. Kekely, V. Puš, and J. Kořenek, "Configurable FPGA Packet Parser for Terabit Networks with Guaranteed Wire-Speed Throughput," in *Proceedings of the 2018 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. ACM, 2018, pp. 249–258, doi: 10.1145/3174243.3174250

[119] J. Li, Z. Sun, and B. Han, "P5: Programmable Parsers with Packet-Level Parallel Processing for FPGA-Based Switches," in *2017 ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)*. IEEE, 2017, pp. 107–108, doi: 10.1109/ANCS.2017.24

[120] Z. Qian and M. Margala, "Low power RAM-based hierarchical CAM on FPGA," in *ReConFigurable Computing and FPGAs (ReConFig), 2014 International Conference on*. IEEE, 2014, pp. 1–4, doi: 10.1109/reconfig.2014.7032536

[121] K. Pagiamtzis and A. Sheikholeslami, "Content-Addressable Memory (CAM) Circuits and Architectures: A Tutorial and Survey," *IEEE Journal of Solid-State Circuits*, vol. 41, no. 3, pp. 712–727, 2006, doi: 10.1109/jssc.2005.864128

[122] W. Jiang, "Scalable Ternary Content Addressable Memory implementation using FPGAs," in *Proceedings of the ninth ACM/IEEE symposium on Architectures for networking and communications systems*. IEEE Press, 2013, pp. 71–82, doi: 10.1109/ancs.2013.6665177

[123] C. A. Zerbini and J. M. Finochietto, "Performance Evaluation of Packet Classification on FPGA-based TCAM Emulation Architectures," in *Global Communications Conference (GLOBECOM), 2012 IEEE*. IEEE, 2012, pp. 2766–2771, doi: 10.1109/glocom.2012.6503535

[124] M. J. Lyons and D. Brooks, "The Design of a Bloom Filter Hardware Accelerator for Ultra Low Power Systems," in *Proceedings of the 2009 ACM/IEEE international symposium on Low power electronics and design*. ACM, 2009, pp. 371–376, doi: 10.1145/1594233.1594330

[125] A. Technology, "ANIC Features Overview," https://accoladetechnology.com/whitepapers/ANIC-Features-Overview.pdf, Tech. Rep., 2017, accessed: 2017-08-02.

[126] M. Molina, S. Niccolini, and N. Duffield, "A Comparative Experimental Study of Hash Functions Applied to Packet Sampling," in *International Teletraffic Congress (ITC-19), Beijing*, 2005.

[127] W. Shi, M. H. MacGregor, and P. Gburzynski, "An Adaptive Load Balancer for Multiprocessor Routers," *Simulation*, vol. 82, no. 3, pp. 173–192, 2006, doi: 10.1177/0037549706067079

[128] "Hash Collision Probabilities," http://preshing.com/20110504/hash-collision-probabilities/, accessed: 2017-07-19.

[129] Z. István, G. Alonso, M. Blott, and K. Vissers, "A Hash Table for Line-Rate Data Processing," *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 8, no. 2, p. 13, 2015, doi: 10.1145/2629582

[130] M. Ahmadi and S. Wong, "Hashing Functions Performance in Packet Classification," in *Proceedings of the International Conference on the Latest Advances in Networks (ICLAN07)*, 2007, pp. 127–132.

[131] A. Fiessler, D. Loebenberger, S. Hager, and B. Scheuermann, "On the Use of (Non-) Cryptographic Hashes on FPGAs," in *International Symposium on Applied Reconfigurable Computing*. Springer, 2017, pp. 72–80, doi: 10.1007/978-3-319-56258-2_7

[132] S. Pati, R. Narayanan, G. Memik, A. Choudhary, and J. Zambreno, "Design and Implementation of an FPGA Architecture for High-Speed Network Feature Extraction," in *Field-Programmable Technology, 2007. ICFPT 2007. International Conference on*. IEEE, 2007, pp. 49–56, doi: 10.1109/fpt.2007.4439231

[133] M. Ruiz, G. Sutter, S. López-Buedo, and J. E. López de Vergara, "FPGA-based Encrypted Network Traffic Identification at 100 Gbit/s," in *ReConFigurable Computing and FPGAs (ReConFig), 2016 International Conference on*. IEEE, 2016, doi: 10.1109/reconfig.2016.7857172

[134] D. D. Clark, V. Jacobson, J. Romkey, and H. Salwen, "An Analysis of TCP Processing Overhead," *IEEE Communications magazine*, vol. 27, no. 6, pp. 23–29, 1989, doi: 10.1109/lcn.1988.10239

[135] J. Postel *et al.*, "Transmission Control Protocol RFC 793," 1981.

[136] N. W. Group *et al.*, "RFC 1071Computing the Internet Checksum, 1988."

[137] J. H. Huang and C.-W. Chen, "On Performance Measurements of TCP/IP and its Device Driver," in *Local Computer Networks, 1992. Proceedings., 17th Conference on*. IEEE, 1992, pp. 568–575, doi: 10.1109/lcn.1992.228142

[138] G. Regnier, S. Makineni, R. Illikkal, R. Iyer, D. Minturn, R. Huggahalli, D. Newell, L. Cline, and A. Foong, "TCP Onloading for Data Center Servers," *Computer*, no. 11, pp. 48–58, 2004, doi: 10.1109/mc.2004.223

[139] J. Kay and J. Pasquale, "Profiling and Reducing Processing Overheads in TCP/IP," *IEEE/ACM Transactions on Networking (TON)*, vol. 4, no. 6, pp. 817–828, 1996, doi: 10.1109/90.556340

[140] J. Touch and B. Parham, "Implementing the Internet Checksum in Hardware," Network Working Group, Tech. Rep., 1996, doi: 10.17487/rfc1936

[141] T. Henriksson, N. Persson, and D. Liu, "VLSI Implementation of Internet Checksum Calculation for 10 Gigabit Ethernet," *Proceedings of Design and Diganostics of Electronics, Cricuits and Systems*, pp. 114–121, 2002.

[142] E. B. Eyo and T. A. Nwodoh, "Designing TCP/IP Checksum Function for Acceleration in FPGA," *Nigerian Journal of Technology*, vol. 29, no. 3, pp. 31–41, 2010.

[143] Atomic Rules., "10/25/40/50/100/400 GbE UDP Offload Engine," Atomic Rules, Tech. Rep., 2017. [Online]. Available: http://www.atomicrules.com/wp-content/uploads/2016/04/AtomicRules_UOE_170822.pdf

[144] D. Sidler, Z. István, and G. Alonso, "Low-latency TCP/IP Stack for Data Center Applications," in *Field Programmable Logic and Applications (FPL), 2016 26th International Conference on*. IEEE, 2016, pp. 1–4, doi: 10.1109/fpl.2016.7577319

[145] N. Zilberman, Y. Audzevich, G. A. Covington, and A. W. Moore, "NetFPGA SUME: Toward 100 Gbps as Research Commodity," *IEEE micro*, vol. 34, no. 5, pp. 32–41, 2014, doi: 10.1109/mm.2014.61

[146] Internet Society. State of IPv6 Deployment 2018. Accessed: 2019-08-08. [Online]. Available: https://www.internetsociety.org/resources/2018/state-of-ipv6-deployment-2018/

[147] J.-P. Deschamps, G. J. Bioul, and G. D. Sutter, *Synthesis of Arithmetic Circuits: FPGA, ASIC and Embedded Systems*. John Wiley & Sons, 2006.

[148] G. Sutter, M. Ruiz, S. López-Buedo, and G. Alonso, "FPGA-based TCP/IP Checksum Offloading Engine for 100 Gbps Networks," in *2018 International Conference on ReConFigurable Computing and FPGAs (ReConFig)*. IEEE, Dec 2018, pp. 1–6, doi: 10.1109/RECONFIG.2018.8641729

[149] "Efficient Checksum Offload Engine," https://github.com/hpcn-uam/efficient_checksum-offload-engine.

[150] Y. Tokusashi, H. T. Dang, F. Pedone, R. Soulé, and N. Zilberman, "The Case For In-Network Computing On Demand," in *Proceedings of the Fourteenth EuroSys Conference 2019*, ser. EuroSys '19. New York, NY, USA: ACM, 2019, pp. 21:1–21:16, doi: 10.1145/3302424.3303979

[151] Y. Hoskote, B. A. Bloechel, G. E. Dermer, V. Erraguntla, D. Finan, J. Howard, D. Klowden, S. G. Narendra, G. Ruhl, J. W. Tschanz *et al.*, "A TCP Offload Accelerator for 10 Gb/s Ethernet in 90-nm CMOS," *IEEE Journal of Solid-State Circuits*, vol. 38, no. 11, pp. 1866–1875, 2003, doi: 10.1109/jssc.2003.818294

[152] H. Jang, S.-H. Chung, D. K. Kim, and Y.-S. Lee, "An Efficient Architecture for a TCP Offload Engine Based on Hardware/Software Co-design." *J. Inf. Sci. Eng.*, vol. 27, no. 2, pp. 493–509, 2011, doi: 10.6688/JISE.2011.27.2.7

[153] Y. Ji and Q.-S. Hu, "40Gbps Multi-Connection TCP/IP Offload Engine," in *Wireless Communications and Signal Processing (WCSP), 2011 International Conference on*. IEEE, 2011, pp. 1–5, doi: 10.1109/wcsp.2011.6096913

[154] A. P. Foong, T. R. Huff, H. H. Hum, J. R. Patwardhan, and G. J. Regnier, "TCP Performance Re-Visited," in *2003 IEEE International Symposium on Performance Analysis of Systems and Software. ISPASS 2003.* IEEE, 2003, pp. 70–79, doi: 10.1109/ispass.2003.1190234

[155] A. Caulfield, E. Chung, A. Putnam, H. Angepat, J. Fowers, M. Haselman, S. Heil, M. Humphrey, P. Kaur, J.-Y. Kim, D. Lo, T. Massengill, K. Ovtcharov, M. Papamichael, L. Woods, S. Lanka, D. Chiou, and D. Burger, "A Cloud-Scale Acceleration Architecture," in *The 49th Annual IEEE/ACM International Symposium on Microarchitecture*, October 2016, doi: 10.1109/micro.2016.7783710

[156] A. Putnam, A. Caulfield, E. Chung, D. Chiou, K. Constantinides, J. Demme, H. Esmaeilzadeh, J. Fowers, J. Gray, M. Haselman, S. Hauck, S. Heil, A. Hormati, J.-Y. Kim, S. Lanka, E. Peterson, A. Smith, J. Thong, P. Y. Xiao, D. Burger, J. Larus, G. P. Gopal, and S. Pope, "A Reconfigurable Fabric for Accelerating Large-Scale Datacenter Services," in *Proceeding of the 41st Annual International Symposium on Computer Architecuture*, vol. 42, no. 3. ACM, 2014, pp. 13–24, doi: 10.1145/2678373.2665678

[157] D. Firestone, A. Putnam, S. Mundkur, D. Chiou *et al.*, "Azure Accelerated Networking: SmartNICs in the Public Cloud," in *NSDI'18)*, 2018.

[158] B. Li, Z. Ruan, W. Xiao, Y. Lu, Y. Xiong, A. Putnam, E. Chen, and L. Zhang, "KV-Direct: High-Performance In-Memory Key-Value Store with Programmable NIC," in *SOSP '17*, 2017, doi: 10.1145/3132747.3132756

[159] E. Chung, J. Fowers, K. Ovtcharov, M. Papamichael, A. Caulfield, T. Massengill, M. Liu, M. Ghandi, D. Lo, S. Reinhardt, S. Alkalay, H. Angepat, D. Chiou, A. Forin, D. Burger, L. Woods, G. Weisz, M. Haselman, and D. Zhang, "Serving DNNs in Real Time at Datacenter Scale with Project Brainwave," *IEEE Micro*, vol. 38, pp. 8–20, March 2018, doi: 10.1109/mm.2018.022071131

[160] Z. István, D. Sidler, G. Alonso, and M. Vukolic, "Consensus in a Box: Inexpensive Coordination in Hardware," in *13th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 16)*. USENIX Association, 2016, pp. 425–438.

[161] D. Sidler, Z. István, M. Owaida, and G. Alonso, "Accelerating Pattern Matching Queries in Hybrid CPU-FPGA Architectures," in *Proceedings of the 2017 ACM International Conference on Management of Data*. ACM, 2017, pp. 403–415, doi: 10.1145/3035918.3035954

[162] A. Sapio, I. Abdelaziz, A. Aldilaijan, M. Canini, and P. Kalnis, "In-Network Computation is a Dumb Idea Whose Time Has Come," in *Proceedings of the 16th ACM Workshop on Hot Topics in Networks*. ACM, 2017, pp. 150–156, doi: 10.1145/3152434.3152461

[163] J. D. Day and H. Zimmermann, "The OSI Reference Model," *Proceedings of the IEEE*, vol. 71, no. 12, pp. 1334–1340, 1983, doi: 10.1109/PROC.1983.12775

[164] K. R. Fall and W. R. Stevens, *TCP/IP Illustrated, Volume 1: The Protocols*. addison-Wesley, 2011.

[165] D. Borman, B. Braden, V. Jacobson, and R. Scheffenegger, "RFC7323: TCP Extensions for High Performance," Internet Engineering Task Force (IETF), Tech. Rep., 2014, doi: 10.17487/rfc7323

[166] Xilinx Inc., "Exact Match Binary CAM Search IP for SDNet," Xilinx Inc., Tech. Rep., 11 2017. [Online]. Available: https://www.xilinx.com/support/documentation/ip_documentation/cam/pg189-cam.pdf

[167] M. Wissolik, D. Zacher, A. Torza, and B. Day, "Virtex UltraScale+ HBM FPGA: A Revolutionary Increase in Memory Performance," Xilinx Inc., Tech. Rep., 07 2019. [Online]. Available: https://www.xilinx.com/support/documentation/white_papers/wp485-hbm.pdf

[168] A. Currid, "TCP Offload to the Rescue," *Queue*, vol. 2, no. 3, pp. 58–65, 2004, doi: 10.1145/1005062.1005069

[169] S. Senapathi and R. Hernandez, "TCP Offload Engines," *Network AND Communications magazine pp103-107*, 2004.

[170] W.-c. Feng, P. Balaji, C. Baron, L. N. Bhuyan, and D. K. Panda, "Performance Characterization of a 10-Gigabit Ethernet TOE," in *High Performance Interconnects, 2005. Proceedings. 13th Symposium on*. IEEE, 2005, pp. 58–63, doi: 10.1109/conect.2005.30

[171] LDA Technologies, "LDA Lightspeed TCP," http://www.ldatech.com/lda-lightspeed-tcp/, accessed: 2018-07-17.

[172] Chevin Technologies, "Chevin Technology's TCP/IP," https://chevintechnology.com/ethernet-ip-2/ct1008-xgtcp/, accessed: 2019-01-09.

[173] Enyx, "10G TCP/IP Full-Hardware Stack IP Core Offload Engine for Xilinx FPGA," http://www.enyx.com/nxtcp-xilinx/, accessed: 2018-07-17.

[174] Enyx, "Enyx Premieres 25G TCP and UDP Offload Engines with Xilinx Virtex UltraScale+ 16nm FPGA on BittWares XUPP3R PCIe Board," http://www.enyx.com/2016/11/enyx-premieres-25g-tcp-udp-offload-engines-wxilinx-virtex-utlrascale-16nm-fpga-bittwares-xupp3r-pcie-board/, accessed: 2018-07-17.

[175] Dini Group, "TCP Offload Engine IP - 128 Sessions (TOE128)," https://www.dinigroup.com/web/TOE128.php, accessed: 2018-07-17.

[176] Algo-Logic, "10G TCP Endpoint," http://algo-logic.com/tcp, accessed: 2018-07-17.

[177] U. Langenbach, A. Berthe, B. Traskov, S. Weide, K. Hofmann, and P. Gregorius, "A 10 GbE TCP/IP Hardware Stack as Part of a Protocol Acceleration Platform," in *2013 IEEE Third International Conference on Consumer Electronics, Berlin (ICCE-Berlin)*. IEEE, 2013, pp. 381–384, doi: 10.1109/ICCE-Berlin.2013.6697997

[178] L. Ding, P. Kang, W. Yin, and L. Wang, "Hardware TCP Offload Engine based on 10-Gbps Ethernet for low-latency Network Communication," in *2016 International Conference on Field-Programmable Technology (FPT)*. IEEE, 2016, pp. 269–272, doi: 10.1109/fpt.2016.7929550

[179] G. Bianchi, M. Welzl, A. Tulumello, G. Belocchi, M. Faltelli, and S. Pontarelli, "A Fully Portable TCP Implementation Using XFSMs," in *Proceedings of the ACM SIGCOMM 2018 Conference on Posters and Demos*. ACM, 2018, pp. 99–101, doi: 10.1145/3234200.3234237

[180] G. Carneiro, "Ns-3: Network simulator 3," in *UTM Lab Meeting April*, vol. 20, 2010, pp. 4–5.

[181] G. Bauer, T. Bawej, U. Behrens, J. Branson, O. Chaze, S. Cittolin, J. A. Coarasa, G.-L. Darlea, C. Deldicque, M. Dobson *et al.*, "10 Gbps TCP/IP Streams from the FPGA for High Energy Physics," in *Journal of Physics: Conference Series*, vol. 513, no. 1. IOP Publishing, 2014, p. 012042, doi: 10.1088/1742-6596/513/1/012042

[182] C. Neely, G. Brebner, and W. Shang, "Flexible and Modular Support for Timing Functions in High Performance Networking Acceleration," in *2010 20th International Conference on Field Programmable Logic and Applications (FPL)*. IEEE, 2010, pp. 513–518, doi: 10.1109/fpl.2010.102

[183] R. Braden, "Requirements for Internet Hosts-Communication Layers," Internet Engineering Task Force, Tech. Rep., 1989, doi: 10.17487/rfc1122

[184] "Count Lines of Code (cloc)," https://github.com/AlDanial/cloc/.

[185] M. Ruiz, D. Sidler, G. Sutter, G. Alonso, and S. López-Buedo, "Limago: an FPGA-based Open-source 100 GbE TCP/IP Stack," in *2019 29th International Confer-*

ence on Field Programmable Logic and Applications (FPL). IEEE, Sep 2019, doi: 10.1109/FPL.2019.00053

[186] "100G-fpga-network-stack-core," https://github.com/hpcn-uam/100G-fpga-network-stack-core/.

[187] L. Woods, Z. István, and G. Alonso, "Ibex - An Intelligent Storage Engine with Support for Advanced SQL Offloading," *Proceedings of the VLDB Endowment*, vol. 7, no. 11, pp. 963–974, 2014, doi: 10.14778/2732967.2732972

[188] L. Woods, J. Teubner, and G. Alonso, "Complex Event Detection at Wire Speed with FPGAs," *Proceedings of the VLDB Endowment*, vol. 3, no. 1-2, pp. 660–669, 2010, doi: 10.14778/1920841.1920926

[189] J. E. López de Vergara, M. Ruiz, L. Gifre, M. Ruiz, L. Vaquero, J. F. Zazo, S. López-Buedo, Ó. González de Dios, and L. Velasco, "Demonstration of 100 Gbit/s Active Measurements in Dynamically Provisioned Optical Paths," in *2019 European Conference on Optical Communication (ECOC)*, 2019.

[190] M. Ruiz, G. Sutter, S. López-Buedo, J. F. Zazo, and J. E. L. de Vergara, "An FPGA-Based Approach for Packet Deduplication in 100 Gigabit-per-Second Networks," in *2017 International Conference on ReConFigurable Computing and FPGAs (ReConFig)*. IEEE, 2017, pp. 1–6, doi: 10.1109/RECONFIG.2017.8279776

[191] M. Ruiz, G. Sutter, T. Alonso, and G. Alonso, "FPGA Efficient Checksum Computation for Multi-Gigabits per Second Networks," *Jornadas Sarteco*, 2018.

[192] "Limago: an FPGA-based Open-source 100 GbE TCP/IP Stack," https://github.com/hpcn-uam/Limago.